НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

"На правах рукопису"
УДК_____

«До захисту допущено»
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
"___"_____2018р.

Магістерська дисертація

зі спеціальності - 122 Комп'ютерні науки та інформаційні технологій
за спеціалізацією - Геометричне моделювання в інформаційних системах
на тему: Система авторизації мікросервісів на основі KeyCloak для захисту середовищ хмарних обчислень

Виконав (-ла): студент (-ка) 6 курсу, групи ТР-71мп
Прижков Артем Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н., доцент Смаковський.Д.С. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.ф.-м.н., доцент. О.В. Галкін _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2018

Національний технічний університет України
“ Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий, магістерський

зі спеціальності - 122 Комп’ютерні науки та інформаційні технології
за спеціалізацією - Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)
«____» _____ 2018р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Прижков Артем Олександрович
(прізвище, ім’я, по батькові)

1. Тема дисертації Система авторизації мікросервісів на основі KeyCloak для захисту середовищ хмарних обчислень

Науковий керівник к.т.н., доцент Смаковський Д.С.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ ” _____201 року №

2. Строк подання студентом дисертації “ ” _____201 року

3. Об’єкт дослідження Проблема авторизація користувачів у системах хмарних обчислень з мікросервісною архітектурою

4. Предмет дослідження Авторизація користувачів у системах хмарних обчислень з мікросервісною архітектурою.

5. Перелік питань, які потрібно розробити
1) проаналізувати сучасні методи авторизації користувачів;
2) проаналізувати методи авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою;
3) розробити архітектуру вирішення поставленої проблеми;
4) розробити програмне забезпечення.

6. Орієнтований перелік ілюстративного матеріалу
мета, постановка задачі, методи авторизації користувачів, , методи авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою;
розробка фреймворку вирішення поставленої проблеми

7. Орієнтований перелік публікацій _____

1) Прижков А.О. Захист середи хмарних обчислень шляхом верифікації програмного забезпечення на наявність деструктивних властивостей / Прижков А.О. Смаковський Д.С. // Сучасні проблеми наукового забезпечення енергетики: Матеріали XVI Міжнародної науково-практичної конференції аспірантів, магістрантів і студентів, м. Київ, 24–27 квітня 2018 р. У 2 т. – К. : 7 КПІ ім. Ігоря Сікорського», 2018. – Т. 2. – 147с.

2) Прижков А.О. Система авторизації мікросервісів на основі KeyCloak для захисту середовищ хмарних обчислень / Прижков А.О. Смаковський Д.С. // Сталий розвиток – XXI століття: управління, технології, моделі Дискусії 2018; колективна монографія, м.Київ, 2018р

8. Дата видачі завдання « 13.09.2017» .

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Строки виконання етапів магістерської дисертації	Приміпка
1	Отримання завдання	13.09.17	
2	Збір інформації	13.09.17 – 1.01.18	
3	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	1.01.18 - 1.06.18	
4	Підготовка матеріалів магістерської роботи	13.09.18 - 24.10.18	
5	Підготовка публікацій	24.04.18 - 27.04.18	
6	Написання основних розділів автореферату	28.10.18 – 26.11.18	
7	Захист програмного продукту	24.10.18	
8	Передзахист	26.11.18	
9	Захист	17.12.18	

Студент _____ Прижков А.О.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Смаковський Д.С.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дисертація складається зі вступу, п’яти розділів, висновку, переліку посилань з 35 найменувань, 2 додатки, і містить 26 рисунків, 22 таблиці. Повний обсяг магістерської дисертації складає 100 сторінки, з яких перелік посилань займає 4 сторінок, додатки – 14 сторінок.

Актуальність теми: Мікросервісна архітектура здобуває все більшу популярність, адже він надає ряд переваг порівняно з монолітною архітектурою, а саме: підтримувати невеликі сервіси легше; можливість горизонтального масштабування; відмовостійкість коду; незалежність вибору технологій для кожного з сервісів. Проте, цей підхід має один суттєвий недолік, для середовищ хмарних обчислень, адже для них зазвичай використовують аутентифікацію по ключам-доступам (наприклад, найпопулярніше хмарне середовище Amason Web Service використовує саме цей тип), а для мікросервісів необхідно використовувати авторизацію користувача лише по токенах, адже потрібно передавати цей токен між усіма мікросервісами. Отже, постає проблема, що нам потрібно передавати два типи унікальних даних, що збільшує розмір запиту і при великій кількості запитів, це може спричинити відмову системи, а також, в обох цих засобах аутентифікації користувачів використовується їх логін та пароль, що є дубліюванням даних.

Метою дослідження є створення фреймворку для вирішення поставленої задачі.

Для досягнення поставленої задачі були сформульовані наступні **завдання дослідження**, що визначили логіку дослідження та його структуру:

- проаналізувати сучасні методи авторизації користувачів;
- проаналізувати сучасні методи моделювання PH-кривих, кривих Без’є, фундаментальних сплайнів та кривих у дробово-раціональному вигляді;
- проаналізувати методи авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою;
- розробити архітектуру вирішення поставленої проблеми;
- розробити програмне забезпечення.

Об’єктом дослідження є проблема авторизація користувачів у системах хмарних обчислень з мікросервісною архітектурою.

Предметом дослідження є авторизація користувачів у системах хмарних обчислень з мікросервісною архітектурою.

Методи дослідження. Розв’язання поставлених задач виконувались з використанням наступних методів:

- методи авторизації користувачів;
- методи авторизації користувачів у системах з мікросервісною архітектурою;
- методи авторизації користувачів у системах хмарних обчислень.

Наукова новизна одержаних результатів. Найбільш суттєвими науковими результатами магістерської дисертації є вирішення проблеми авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою.

Практичне значення одержаних результатів роботи полягає в розробка фреймворку, що вирішує проблему авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою.

Ключові слова. *АВТОРИЗАЦІЯ, СИСТЕМИ ХМАРНИХ ОБЧИСЛЕНЬ, МІКРОСЕРВІСНА АРХІТЕКТУРА, АВТОРИЗАЦІЯ ПО ТОКЕНАХ6 АВТОРИЗАЦІЯ ПО КЛЮЧУ-ДОСТУПУ.*

ABSTRACT

Master's thesis consists of an introduction, five sections, a conclusion, a list of references from 35 denominations, 2 appendices, and have 26 figures, 22 tables. The full volume of the master's thesis is 100 pages, 4 of which is list of references, 25 – appendices.

Topicality. Microservice architecture is gaining in popularity, since it offers a number of advantages over a monolithic architecture, namely: to support small services more easily; the possibility of horizontal scaling; fault tolerance code; Independence of the choice of technologies for each of the services. However, this approach has one significant drawback for cloud computing environments, since they typically use authentication for access keys (for example, the most popular cloud environment uses this type of Amason Web Service), and for microservices it is necessary to use user authorization only by tokens, After all, it is necessary to pass this token between all microservices. So, there is a problem that we need to transfer two types of unique data, which increases the size of the request and with a large number of requests, this can lead to system failure, and in both of these authentication tools, users use their login and password, which is a duplication of data..

The aim of the research is to create a framework for solving the problem.

To accomplish the task, the following **research objectives** were formulated, which determined the logic of the research and its structure:

- analyze modern methods of user authentication;
- analyze modern methods of modeling PH-curves, Bezier curves, fundamental splines and curves in fractional-rational form;
- analyze user authorization methods in cloud computing systems with micro-server architecture;
- to develop the architecture of the solution of the problem;
- develop software.

The object of research is a problem authorizing users in cloud computing systems with the micro-server architecture.

The subject of research is the authorization of users in cloud computing systems with the micro-server architecture.

Research Methods. The solving of defined tasks was performed using the following methods:

- Authorization methods of users;
- Authorization methods for users in systems with micro-service architecture;
- Authorization methods for users in cloud computing systems.

Scientific novelty. The most significant scientific results of the master's thesis are solving the problem of authorizing users in cloud computing systems with micro-server architecture.

The practical value of research is to develop a framework that solves the problem of user authorization in cloud computing systems with micro-server architecture ...

Keywords. *AUTHORIZATION, SYSTEMS OF CHROME COMPUTERS, MICRO-SERVICE ARCHITECTURE, AUTHORIZATION ON TOXEN6 AUTHORIZATION BY KEY-ACCESS.*

ЗМІСТ

ВСТУП

Термін “мікросервісна архітектура” з’явився протягом останніх кількох років, щоб описати спосіб проектування програмних застосувань як наборів самостійно розгорнутих сервісів. Хоча точного визначення цього архітектурного стилю не існує, існують певні загальні характеристики: організації сервісів навколо бізнес потреб, автоматичне розгортання, перенесення логіки від шини повідомлень до приймачів, децентралізований контроль над мовами програмування та даними». (М. Фаулер) [1].

При розробці програмного забезпечення першим питанням постає тип архітектури застосунку. Наразі існує два підходи до архітектури застосунків — моноліт та мікросервіси. Монолітний додаток будується як єдине ціле. Прикладні програми підприємства складаються з трьох частин: бази даних (що складається з багатьох таблиць, як правило, в реляційній системі управління базами даних), клієнтського інтерфейсу користувача (що складається з HTML-сторінок та / або JavaScript коду, запущених у веб-переглядачі), а також на серверної стороні. Ця прикладна програма на сервері буде обробляти HTTP-запити, виконувати певну логіку для домену, завантажувати та оновлювати дані з бази даних, а також заповнювати HTML-представлення, що надсилаються в браузер. Отже, моноліт — це єдиний логічний виконуваний файл. Щоб внести будь-які зміни в систему, розробник повинен побудувати та розгорнути оновлену версію програми на стороні сервера.

На відміну від цього, можливості мікросервісу виражаються формально за допомогою бізнес-орієнтованих API. Вони складають основну ділову спроможність, і тому є цінними активами для бізнесу. Реалізація служби, яка може включати інтеграцію з системами запису, повністю схована, оскільки інтерфейс визначається виключно в комерційних термінах. Позиціонування послуг як цінних активів для бізнесу неявно сприяє їх адаптації до використання в різних контекстах. Одна і та ж послуга може бути використана повторно в декількох бізнес-процесах

або в різних бізнес-каналах або цифрових точках зв'язку, залежно від потреби. Залежність між службами та їх споживачами мінімізуються, застосовуючи принцип вільного з'єднання. За стандартизацією контрактів, виражених через бізнес-орієнтовані API, споживачам не впливають зміни в реалізації послуги. Це дозволяє власникам послуг змінювати реалізацію та модифікувати системи записів або сервісних композицій, які можуть лежати поза інтерфейсом і замінювати їх без будь-якого впливу на нижній рівень [2].

Мікросервісна архітектура має пропонує переваги для бізнесу, що може знизити витрати. Ці переваги включають:

- гнучкість — розбиваючи функціональні можливості на самий базовий рівень, а потім абстрагуючи пов'язані служби, адміністратор може зосередитися лише на оновленні відповідних частин програми. Це усуває болісний процес інтеграції, який зазвичай пов'язаний з монолітними додатками. Микросервісний підхід збільшує швидкість розробки, перетворюючи її в процес, який може бути виконаний тижнями, а не місяцями;
- ефективність — використання архітектури на базі мікросервісів може привести до набагато більш ефективного використання коду та базової інфраструктури. Звичайно, істотна економія витрат становить до 50%, зменшуючи обсяг інфраструктури, необхідної для запуску даної заявки;
- відновлюваність — розподіл функціональних можливостей по декількох службах усуває сприйнятливість програм до відмови в одній точці. Результати в додатках, які працюють ефективніше, забезпечують менше простоїв і можуть масштабувати за потребою;
- дохід — швидша ітерація та зменшення часу простою може збільшити прибуток. Збереження та залучення користувачів зростає завдяки постійним покращенням мікросервісів [2].

Проте, цей підхід не позбавлений недоліків:

- потрібно опрацьовувати набагато більше програм, ніж просто одину чи дві. Для вирішення даної проблеми використовують технології контейнерів

(наприклад, Docker), PaaS (наприклад Cloud Foundry) та методологію безперебійної доставки для пом'якшення цього недоліку;

- у випадку, якщо ваша архітектура має погано організована, ви помножите всі проблеми, розбиваючи програму на різні мікросервіси [3];
- необхідно використовувати авторизацію користувача лише по токенах, адже потрібно передавати цей токен між усіма мікросервісами, що сворює проблему аутентифікації у середовищах хмарних обчислень, адже в них використовують авторизацію по ключам-доступам. Отже, постає проблема, що нам потрібно передавати два типи унікальних даних, що збільшує розмір запиту і при великій кількості запитів, це може спричинити відмову системи, а також, в обох цих засобах аутентифікації користувачів використовується їх логін та пароль, що є дубліюванням даних.

Метою дослідження є створення фреймворку, що буде вирішувати дану проблему. Для досягнення поставленої задачі були сформульовані наступні завдання дослідження, що визначили логіку дослідження та його структуру:

- проаналізувати сучасні методи аутентифіцкації користувачів;
- проаналізувати формати токенів;
- розробити архітектуру застосунку, що вирішує поставлену проблему авторизації в системах хмарних обчислень з мікросервісною архітектурою;
- розробити фреймворк, що вирішує поставлену задачу.

1. СУЧАСНІ МЕТОДИ АВТОРИЗАЦІЇ КОРИСТУВАЧІВ

На сьогоднішній день існує велика кількість способів авторизації користувачів у системах, проте спочатку потрібно чітко розуміти термінологію.

Ідентифікація — це заява про те, ким ви є. Залежно від ситуації, це може бути ім'я, адреса електронної пошти, номер облікового запису, ітд.

Аутентифікація — надання доказів, що ви насправді є той, ким ідентифікувалися (від слова "authentic" — істинний, справжній) [4].

Авторизація — перевірка, що вам дозволено доступ до запитуваного ресурсу.

Наприклад, при спробі потрапити в закритий клуб вас ідентифікують (запитають ваше ім'я і прізвище), аутентифицируют (попросять показати паспорт і звірять фотографію) і авторизують (перевірять, що прізвище знаходиться в списку гостей), перш ніж пустять усередину [5].

Аналогічно ці терміни вживаються в комп'ютерних системах, де традиційно під ідентифікацією розуміють отримання вашого профілю (identity) по username або email; під аутентифікацією — перевірку, що ви знаєте пароль від цього облікового запису, а під авторизацією - перевірку вашої ролі в системі і рішення про надання доступу до запитаної сторінці або ресурсу.

Роздивимося різні підходи до аутентифікація користувачів:

- аутентифікація по паролю;
- аутентифікація по сертификатам;
- аутентифікація по ключам-доступу;
- аутентифікація по токенах.

1.1. Аутентифікація по паролю

Цей метод грунтується на тому, що користувач повинен надати username і password для успішної ідентифікації і аутентифікації в системі. Пара username / password задається користувачем при його реєстрації в системі, при цьому в якості

username може виступати адреса електронної пошти користувача. Стосовно до веб-додатків, існує кілька стандартних протоколів для аутентифікації по паролю, а саме: HTTP, Forms, URL query, Request body, HTTP header.

Описаний в стандартах HTTP 1.0 / 1.1, HTTP authentification використовується вже багато років і за цей час став корпоритвним стандартом. Розглянемо схему взаємодії з веб-сайтами:

1. Спочатку, неавторизований клієнт робить запит до захищеного ресурсу на сервері, той у свою чергу відповідає HTTP статусом "401 Unauthorized", також у заголовок додається "WWW-Authenticate", у якому міститься схема і параметри аутентифікації;

2. Отримавши таку відповідь, браузер автоматично виводить на екран користувача діалог введення username і password, куди користувач вводить дані свого облікового запису;

3. У подальших викликах на сервер даного захищеного ресурсу, браузер додає у заголовок запиту "Authorization", в якому знаходяться всі дані користувача, які необхідні для аутентифікації сервером;

4. Сервер проводить аутентифікацію користувачів спираючись на дані отримані з цього заголовку та робить рішення допускати до захищенного ресурсу користувача, в залежності від ролі, ACL або інших даних [6].

Весь процес стандартизований і добре підтримується всіма браузерами і веб-серверами. Існує кілька схем аутентифікації, що відрізняються за рівнем безпеки:

- Basic - найбільш проста схема, при якій username і password користувача передаються в заголовку Authorization в незашифрованому вигляді (base64-encoded). Однак при використанні HTTPS протоколу, є відносно безпечною (рисунок 1.1);
- Digest - challenge-response-схема, при якій сервер посилає унікальне значення nonce, а браузер передає MD5 хеш пароля користувача, обчислений з використанням зазначеного nonce. Більш безпечна альтернатівв Basic схеми при незахищених з'єднаннях, але схильна до man-in-the-middle attacks (з заміною схеми

на basic). Крім того, використання цієї схеми не дозволяє застосувати сучасні хеш-функції для зберігання паролів користувачів на сервері;

- NTLM (відома як Windows authentication) - також заснована на challenge-response підході, при якому пароль не передається в чистому вигляді. Ця схема не є стандартом HTTP, але підтримується більшістю браузерів і веб-серверів. Переважно використовується для аутентифікації користувачів Windows Active Directory в веб-додатках. Вразлива до pass-the-hash-атакам;
- Negotiate - ще одна схема з сімейства Windows authentication, яка дозволяє клієнтові вибрати між NTLM і Kerberos аутентифікації. Kerberos - більш безпечний протокол, заснований на принципі Single Sign-On. Однак він може функціонувати, тільки якщо і клієнт, і сервер знаходяться в зоні intranet і є частиною домену Windows.

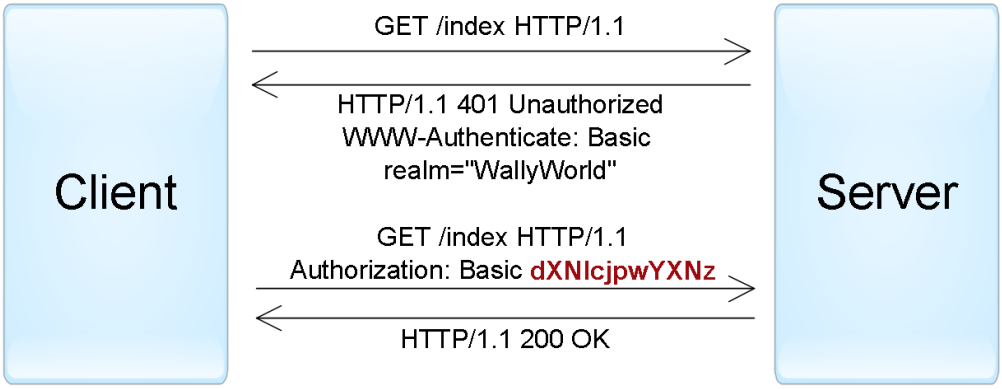

Рисунок 1.1 Схема Basic аутентифікації

Варто відзначити, що при використанні HTTP-аутентифікації у користувача немає стандартної можливості вийти з веб-додатки, крім як закрити всі вікна браузера.

Forms authentification протокол немає певного стандарту, тому всі його реалізації специфічні для конкретних систем, а точніше, для модулів аутентифікації фреймворків розробки. Працює за наступним принципом: в веб-додаток включається HTML-форма, в яку користувач повинен ввести свої username / password і відправити їх на сервер через HTTP POST для аутентифікації. У разі успіху веб-додаток створює session token, який зазвичай поміщається в browser

cookies. При наступних веб-запитах session token автоматично передається на сервер і дозволяє додатком отримати інформацію про поточного користувача для авторизації запиту. (рисунок 1.2)

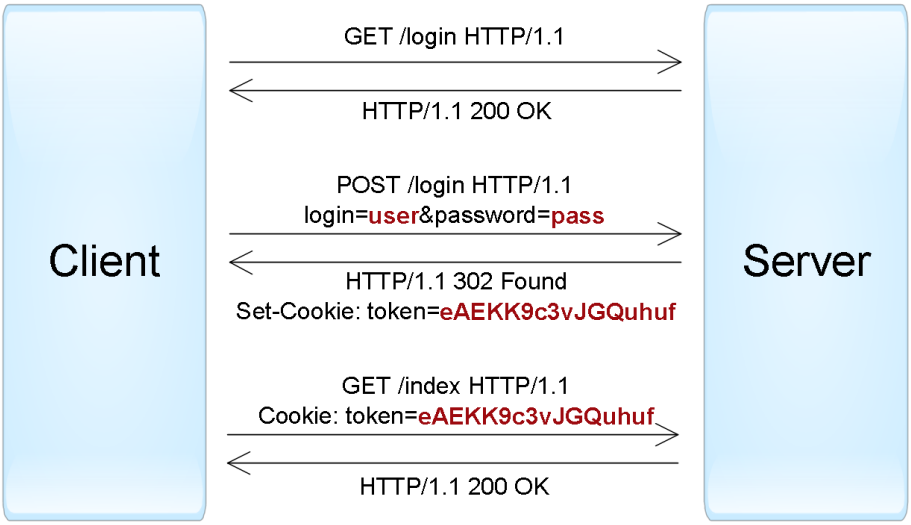

Рисунок 1.2 Схема Forms аутентифікації

Додаток може створити session token двома способами:

1. Як ідентифікатор аутентифицироваться сесії користувача, яка зберігається в пам'яті сервера або в базі даних. Сесія повинна містити всю необхідну інформацію про користувача для можливості авторизації його запитів;

2. Як зашифрований і / або підписаний об'єкт, що містить дані про користувача, а також вказано термін. Цей підхід дозволяє реалізувати stateless-архітектуру сервера, однак вимагає механізму поновлення сесійного токена після закінчення терміну дії. Кілька стандартних форматів таких токенов розглядаються в секції «Аутентифікація по токені».

Необхідно розуміти, що перехоплення session token часто дає аналогічний рівень доступу, що і знання username / password. Тому всі комунікації між клієнтом і сервером у разі forms authentication повинні проводитися тільки по захищеному з'єднанню HTTPS.

Інші протоколи аутентифікації по паролю. Два протоколу, описаних вище, успішно використовуються для аутентифікації користувачів на веб-сайтах. Але при розробці клієнт-серверних додатків з використанням веб-сервісів (наприклад, iOS або Android), поряд з HTTP аутентифікації, часто застосовуються нестандартні

протоколи, в яких дані для аутентифікації передаються в інших частинах запиту. Існує всього декілька місць, де можна передати username і password в HTTP запитах:

- URL query - вважається небезпечним варіантом, тоді як рядки URL можуть запам'ятовуватися браузерами, проксі і веб-серверами;
- Request body - безпечний варіант, але він застосовується лише для запитів, що містять тіло повідомлення (такі як POST, PUT, PATCH);
- HTTP header-оптимальний варіант, при цьому можуть використовуватися і стандартний заголовок Authorization (наприклад, з Basic-схемою), та інші довільні заголовки.

За простотою реалізації даного методу ховається багато недоліків і тому даний спосіб вважаться не дуж надійним. Паролі часто можна підібрати, а користувачі схильні використовувати прості і однакові паролі в різних системах, або записувати їх на клаптиках паперу. Якщо зловмисник зміг з'ясувати пароль, то користувач часто про це не дізнається. Крім того, розробники додатків можуть допустити ряд концептуальних помилок, що спрощують злом облікових записів. Нижче представлений список найбільш часто зустрічаються вразливостей в разі використання аутентифікації по паролю:

- веб-додаток дозволяє користувачам створювати прості паролі;
- веб-додаток не захищене від можливості перебору паролів (brute-force attacks);
- веб-додаток саме генерує і поширює паролі користувачам, однак не вимагає зміни пароля після першого входу (тобто поточний пароль десь записаний);
- веб-додаток допускає передачу паролів по незахищеному HTTP-з'єднання або в рядку URL;
- веб-додаток не використовує безпечні хеш-функції для зберігання паролів користувачів;
- веб-додаток не надає користувачам можливість зміни пароля або не нотифікує користувачів про зміну їх паролів;

- веб-додаток використовує вразливу функцію відновлення пароля, яку можна використовувати для отримання несанкціонованого доступу до інших облікових записів;
- веб-додаток не вимагає повторної аутентифікації користувача для важливих дій: зміна пароля, зміни адреси доставки товарів і т.п.;
- веб-додаток створює session tokens таким чином, що вони можуть бути підібрані або передбачені для інших користувачів;
- веб-додаток допускає передачу session tokens по незахищеному HTTP-з'єднання, або в рядку URL;
- веб-додаток вразливе для session fixation-атак (не замінює session token при переході анонімної сесії користувача в аутентифицироваться);
- веб-додаток не встановлює прапори HttpOnly і Secure для browser cookies, що містять session tokens;
- веб-додаток не знищує сесії користувача після короткого періоду неактивності або не надає функцію виходу з аутентифицироваться сесії.

1.2. Аутентифікація по сертифікатам

Сертифікат являє собою набір атрибутів, що ідентифікують власника, підписаний certificate authority (CA). CA виступає в ролі посередника, який гарантує справжність сертифікатів. Також сертифікат криптографически пов'язаний з закритим ключем, яких зберігається у власника сертифіката і дозволяє однозначно підтвердити факт володіння сертифікатом. На стороні клієнта сертифікат разом з закритим ключем можуть зберігатися в операційній системі, в браузері, в файлі, на окремому фізичному пристрої (smart card, USB token). Зазвичай закритий ключ додатково захищений паролем або PIN-кодом. У веб-додатках традиційно використовують сертифікати стандарту X.509. Аутентифікація за допомогою X.509-сертифіката відбувається в момент з'єднання з сервером і є частиною протоколу SSL / TLS. Цей механізм також добрепідтримується браузерами, які дозволяють користувачеві вибрати і

застосувати сертифікат, якщо веб-сайт допускає такий спосіб аутентифікації. (рисунок 1.3)

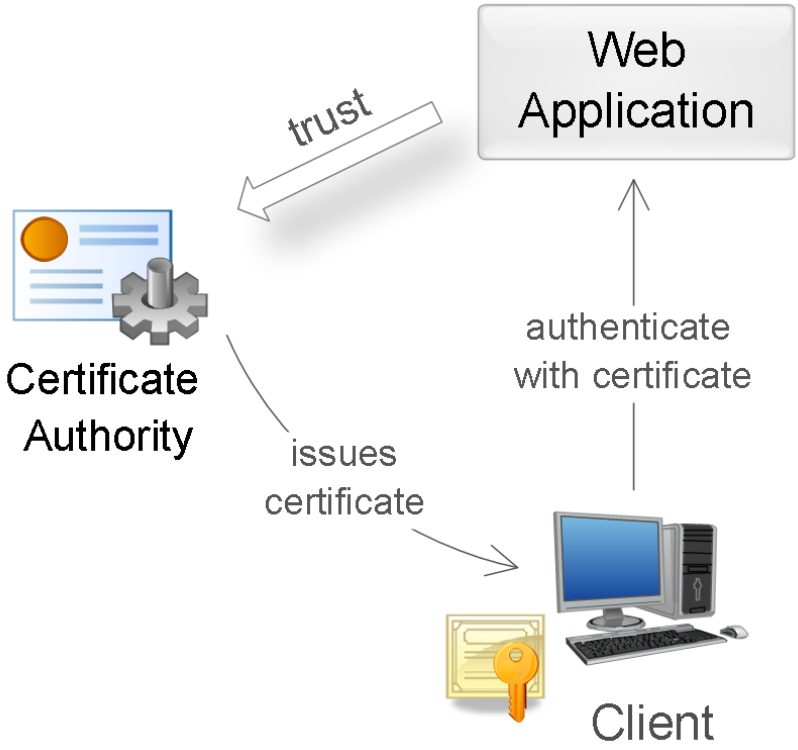

Рисунок 1.3 Схема аутентифікація по сертификатам

Під час аутентифікації сервер виконує перевірку сертифіката на підставі наступних правил:

- Сертифікат повинен бути підписаний довіреною certification authority (перевірка ланцюжка сертифікатів);
- Сертифікат повинен бути дійсним на поточну дату (перевірка терміну дії);
- Сертифікат не повинен бути відкликаний відповідним CA (перевірка списків виключення).

Після успішної аутентифікації веб-додаток може виконати авторизацію запиту на підставі таких даних сертифіката, як subject (ім'я власника), issuer (емітент), serial number (серійний номер сертифіката) або thumbprint (відбиток відкритого ключа сертифіката).

Використання сертифікатів для аутентифікації - куди більш надійний спосіб, ніж аутентифікація за допомогою паролів. Це досягається створенням в процесі аутентифікації цифрового підпису, наявність якої доводить факт застосування

закритого ключа в конкретній ситуації (non-repudiation). Однак труднощі з поширенням і підтримкою сертифікатів робить такий спосіб аутентифікації малодоступним в широких колах.

Аутентифікація за одноразовими паролями зазвичай застосовується додатково до аутентифікації по паролів для реалізації two-factor authentication (2FA). У цій концепції користувачеві необхідно надати дані двох типів для входу в систему: щось, що він знає (наприклад, пароль), і щось, чим він володіє (наприклад, пристрій для генерації одноразових паролів). Наявність двох факторів дозволяє в значній мірі збільшити рівень безпеки, що м. Б. затребуване для певних видів веб-додатків.

Інший популярний сценарій використання одноразових паролів - додаткова аутентифікація користувача під час виконання важливих дій: переказ грошей, зміна налаштувань і т.п. Існують різні джерела для створення одноразових паролів. Найбільш популярні:

- апаратні або програмні маркери, які можуть генерувати одноразові паролі на підставі секретного ключа, введеного в них, і поточного часу. Секретні ключі користувачів, які є фактором володіння, також зберігаються на сервері, що дозволяє виконати перевірку введених одноразових паролів. Приклад апаратної реалізацій токенов - RSA SecurID; програмної - додаток Google Authenticator;
- випадково генеруються коди, що передаються користувачеві через SMS або інший канал зв'язку. У цій ситуації фактор володіння - телефон користувача (точніше - SIM-карта, прив'язана до певного номера);
- роздруківка або scratch card зі списком заздалегідь сформованих одноразових паролів. Для кожного нового входу в систему потрібно ввести новий одноразовий пароль з зазначеним номером.

У веб-додатках такий механізм аутентифікації часто реалізується за допомогою розширення forms authentication: після первинної аутентифікації по паролю, створюється сесія користувача, проте в контексті цієї сесії користувач не має доступу до додатка до тих пір, поки він не виконає додаткову аутентифікацію за одноразовим паролем.

1.3. Аутентифікація по ключам доступу

Цей спосіб найчастіше використовується для аутентифікації пристроїв, сервісів або інших додатків при зверненні до веб-сервісів. Тут в якості секрету застосовуються ключі доступу (access key, API key) - довгі унікальні рядки, що містять довільний набір символів, по суті замінюють собою комбінацію username / password.

У більшості випадків, сервер генерує ключі доступу за запитом користувачів, які далі зберігають ці ключі в клієнтських додатках. При створенні ключа також можливо обмежити термін дії і рівень доступу, який отримає клієнтську програму при аутентифікації за допомогою цього ключа.

Гарний приклад застосування аутентифікації по ключу - хмара Amazon Web Services. Припустимо, у користувача є веб-додаток, що дозволяє завантажувати і переглядати фотографії, і він хоче використовувати сервіс Amazon S3 для зберігання файлів. В такому випадку, користувач через консоль AWS може створити ключ, що має обмежений доступ до хмари: тільки читання / запис його файлів в Amazon S3. Цей ключ в результаті можна застосувати для аутентифікації веб-додатки в хмарі AWS. (рисунок 1.4) [7].

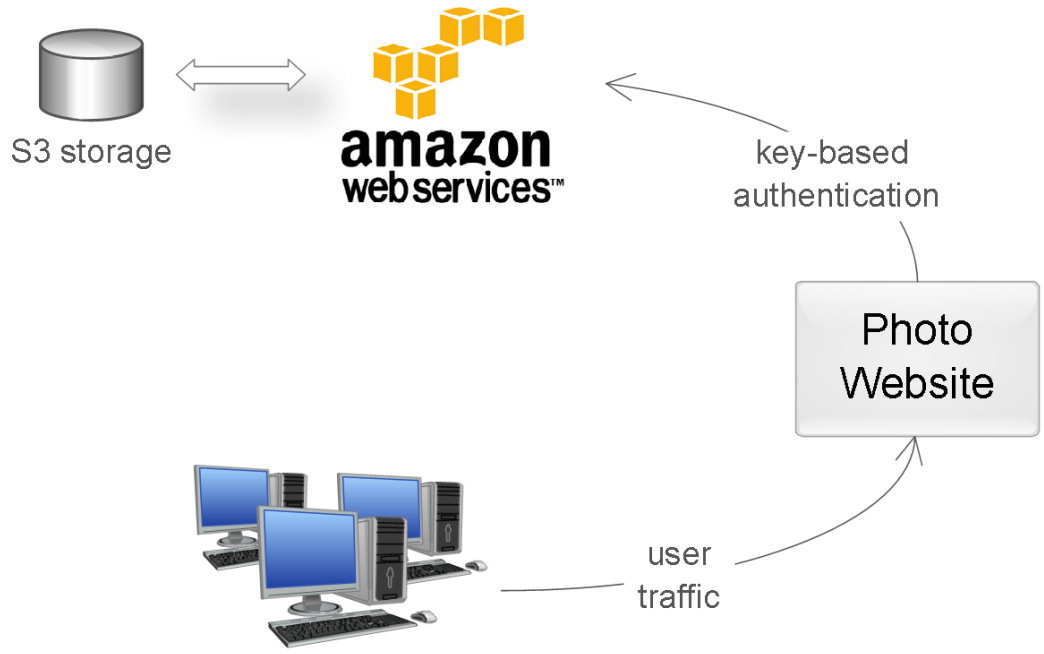

Рисунок 1.4 Схема аутентифікація по ключу

Використання ключів дозволяє уникнути передачі пароля користувача стороннім додаткам (в прикладі вище користувач зберіг в веб-додатку не свій пароль, а ключ доступу). Ключі мають значно більшу ентропією в порівнянні з паролями, тому їх практично неможливо підібрати. Крім того, якщо ключ був розкритий, це не призводить до компрометації основний облікового запису користувача - достатньо лише анулювати цей ключ і створити новий.

З технічної точки зору, тут не існує єдиного протоколу: ключі можуть передаватися в різних частинах HTTP-запиту: URL query, request body або HTTP header. Як і в випадку аутентифікації по паролю, найбільш оптимальний варіант - використання HTTP header. У деяких випадках використовують HTTP-схему Bearer для передачі токена в заголовку (Authorization: Bearer [token]). Щоб уникнути перехоплення ключів, з'єднання з сервером має бути обов'язково захищене протоколом SSL / TLS. (рисунок 1.5)

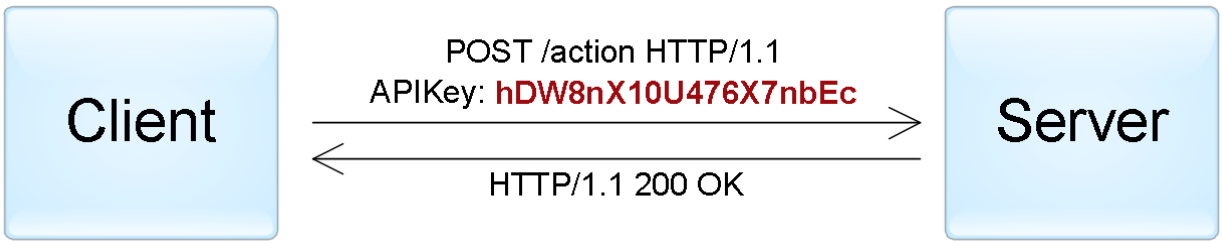

Рисунок 1.5 Схема аутентифікації по ключу доступу, переданого в HTTP заголовку

Крім того, існують більш складні схеми аутентифікації по ключам для незахищених з'єднань. В цьому випадку, ключ зазвичай складається з двох частин: публічної і приватної. Публічна частина використовується для ідентифікації клієнта, а приватна частина дозволяє згенерувати, так званий електронний підпис, що використовується для декодування запиту направленного на сервер. Наприклад, за аналогією з digest authentication схемою, сервер може послати клієнту унікальне значення nonce або timestamp, а клієнт - повернути хеш або HMAC цього значення, обчислений з використанням секретної частини ключа. Це дозволяє уникнути передачі всього ключа в оригінальному вигляді і захищає від атак, що роблять повторні запити на сервер.

1.4. Аутентифікація по токенах

Найчастіше застосовується при побудові розподілених систем Single Sign-On (SSO), де один додаток (service provider або relying party) делегує функцію аутентифікації користувачів іншому додатку (identity provider або authentication service). Типовий приклад цього способу - вхід в додаток через обліковий запис в соціальних мережах. Тут соціальні мережі є сервісами аутентифікації, а додаток довіряє функцію аутентифікації користувачів соціальних мереж.

Реалізація цього способу полягає в тому, що identity provider (IP) надає достовірні відомості про користувача в вигляді токена, а service provider (SP) додаток використовує цей токен для ідентифікації, аутентифікації і авторизації користувача (рисунок 1.6)[8].

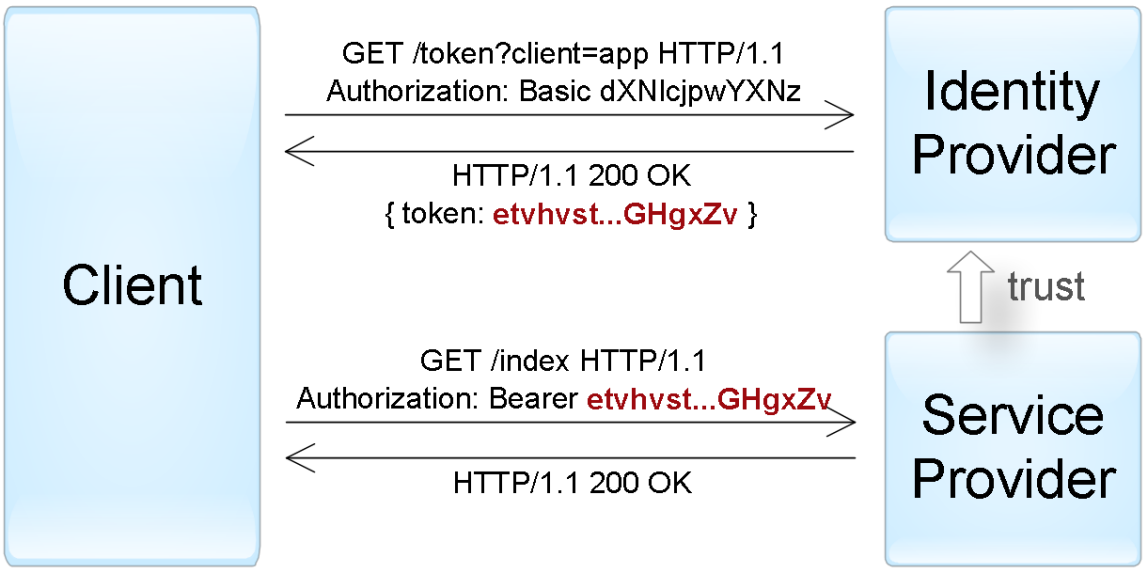

Рисунок 1.6 Схема аутентифікації «активного» клієнта за допомогою токена, переданого за допомогою Bearer схеми.

На загальному рівні, весь процес виглядає наступним чином:

- клієнт аутентифицирующей в identity provider одним із способів, специфічним для нього (пароль, ключ доступу, сертифікат, Kerberos, итд.);
- клієнт просить identity provider надати йому токен для конкретного SP-додатки. Identity provider генерує токен і відправляє його клієнту;

- клієнт аутентифицирующей в SP-додатку за допомогою цього токена. Процес, описаний вище, відображає механізм аутентифікації активного клієнта, тобто такого, який може виконувати запрограмовану послідовність дій (наприклад, iOS / Android програми). Браузер ж — пасивний клієнт в тому сенсі, що він тільки може відображати сторінки, запитані користувачем. В цьому випадку аутентифікація досягається за допомогою автоматичного перенаправлення браузера між веб-додатками identity provider і service provider.

Існує кілька стандартів, в точності що визначають протокол взаємодії між клієнтами (активними і пасивними) і IP / SP-додатками і формат підтримуваних токенов. Серед найбільш популярних стандартів - OAuth, OpenID Connect, SAML, і WS-Federation. (рисунок 1.7)

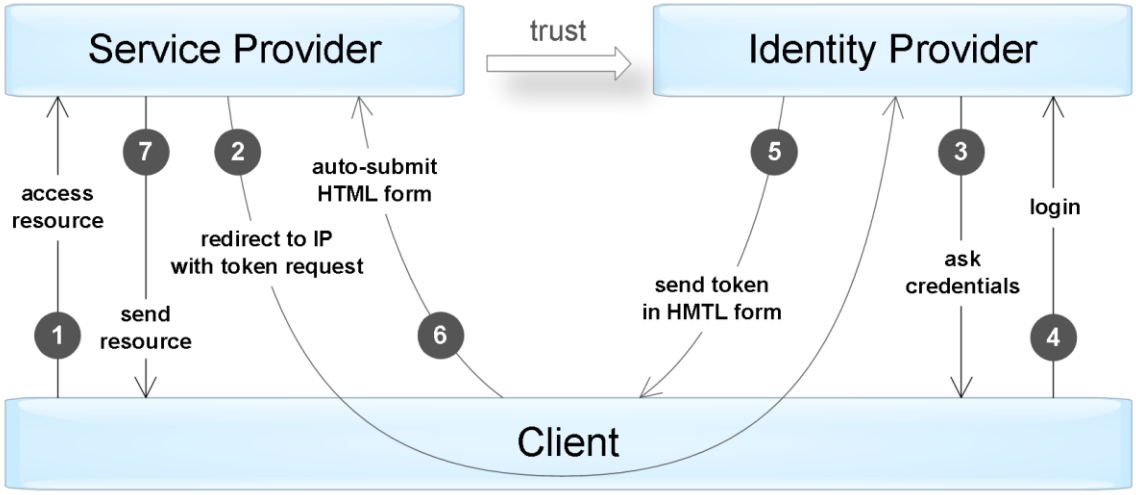

Рисунок 1.7 Схема аутентифікації «пасивного» клієнта за допомогою токена, переданого за допомогою Bearer схеми.

Сам токен зазвичай являє собою структуру даних, яка містить інформацію, хто згенерував токен, хто може бути одержувачем токена, термін дії, набір відомостей про самого користувача (claims). Крім того, токен додатково підписується для запобігання несанкціонованих змін і гарантій автентичності.

При аутентифікації за допомогою токена SP-додаток повинен виконати наступні перевірки:

1. Токен був виданий довіреною identity provider додатком (перевірка поля issuer).

- Токен призначається поточному SP-додатком (перевірка поля audience).
- Термін дії токена ще не закінчився (перевірка поля expiration date).
- Токен справжній і не був змінений (перевірка підпису).

У разі успішної перевірки SP-додаток виконує авторизацію запиту на підставі даних про користувача, що містяться в токені.

1.5. Види форматів токенів

Існує кілька поширених форматів для веб-додатків:

- Simple Web Token (SWT) — найбільш простий формат, який представляє собою набір довільних пар ім'я / значення в форматі кодування HTML form. Стандарт визначає кілька зарезервованих імен: Issuer, Audience, ExpiresOn і HMACSHA256. Токен підписується за допомогою симетричного ключа, таким чином обидва IP- і SP-додатки повинні мати цей ключ для можливості створення / перевірки токена.
- JSON Web Token (JWT) — містить три блоки, між якими ставлять крапку: заголовок, набір полів (claims) і підпис. Перші два блоки представлені в JSON-форматі і додатково закодовані в формат base64. Набір полів містить довільні пари ім'я / значення, до того ж стандарт JWT визначає кілька зарезервованих імен (iss, aud, exp і інші). Підпис може генеруватися за допомогою і симетричних алгоритмів шифрування, і асиметричних. Крім того, існує окремий стандарт, відписувався формат зашифрованого JWT-токена.
- Security Assertion Markup Language (SAML) - визначає токени (SAML assertions) в XML-форматі, що включає інформацію про емітента, про суб'єкта, необхідні умови для перевірки токена, набір додаткових тверджень (statements) про користувача. Підпис SAML-токенів здійснюється за допомогою асиметричної криптографії. Крім того, на відміну від попередніх форматів, SAML-токени містять механізм для підтвердження володіння токеном, що дозволяє запобігти перехоплення токенов через man-in-the-middle-атаки при використанні незахищених з'єднань [9].

Стандарт Security Assertion Markup Language (SAML) описує способи взаємодії і протоколи між identity provider і service provider для обміну даними аутентифікації і авторизації за допомогою токенів. Спочатку версії 1.0 і 1.1 були випущені в 2002 - 2003 рр., В той час як версія 2.0, значно розширює стандарт і назад несумісна, опублікована в 2005 році.

Цей основоположний стандарт - досить складний і підтримує багато різних сценаріїв інтеграції систем. Основні «будівельні блоки» стандарту:

- Assertions - власний формат SAML токенов в XML форматі;
- Protocols - набір підтримуваних повідомлень між учасниками, серед яких - запит на створення нового токена, отримання існуючих токенів, вихід із системи (logout), управління ідентифікаторами користувачів, і інші;
- Bindings - механізми передачі повідомлень через різні транспортні протоколи. Підтримуються такі способи, як HTTP Redirect, HTTP POST, HTTP Artifact (посилання на повідомлення), SAML SOAP, SAML URI (адреса отримання повідомлення) та інші;
- Profiles - типові сценарії використання стандарту, що визначають набір assertions, protocols і bindings необхідних для їх реалізації, що дозволяє досягти кращої сумісності. Web Browser SSO - один із прикладів таких профілів.

Крім того, стандарт визначає формат обміну метаінформацією між учасниками, яка включає список підтримуваних ролей, протоколів, атрибутів, ключі шифрування і т. п. [10].

Також існують такі стандарти як: WS-Trust та WS-Federation, а також OAuth та OpenID Connect.

WS-Trust і WS-Federation входять в групу стандартів WS- *, що описують SOAP / XML-веб сервіси. Ці стандарти розробляються групою компаній, куди входять Microsoft, IBM, VeriSign і інші. Поряд з SAML, ці стандарти досить складні, використовуються переважно в корпоративних сценаріях [9].

Стандарт WS-Trust описує інтерфейс сервісу авторизації, іменованого Secure Token Service (STS). Цей сервіс працює по протоколу SOAP і підтримує створення, оновлення та анулювання токенов. При цьому стандарт допускає

використання токенов різного формату, проте на практиці в основному використовуються SAML-токени.

Стандарт WS-Federation стосується механізмів взаємодії сервісів між компаніями, зокрема, протоколів обміну токенов. При цьому WS-Federation розширює функції та інтерфейс сервісу STS, описаного в стандарті WS-Trust. Серед іншого, стандарт WS-Federation визначає:

- формат і способи обміну метаданими про сервіси;
- функцію єдиного виходу з усіх систем (single sign-out);
- сервіс атрибутів, що надає додаткову інформацію про користувача;
- сервіс псевдонімів, що дозволяє створювати альтернативні імена користувачів;
- підтримку пасивних клієнтів (браузерів) за допомогою перенаправлення.

Можна сказати, що WS-Federation дозволяє вирішити ті ж завдання, що і SAML, проте їх підходи і реалізація в деякій мірі відрізняються.

На відміну від SAML і WS-Federation, стандарт OAuth (Open Authorization) не описує протокол автентифікації користувача. Замість цього він визначає механізм отримання доступу однієї програми до іншого за імені користувача. Однак існують схеми, що дозволяють здійснити аутентифікацію користувача на базі цього стандарту.

Перша версія стандарту розроблена в 2007 - 2010 рр .., а поточна версія 2.0 опублікована в 2012 р. Версія 2.0 значно розширює і в той же час спрощує стандарт, але зворотна несумісна з версією 1.0. Сьогодні OAuth 2.0 дуже популярний і використовується повсюдно для надання делегованого доступу та третього стороннього аутентифікації користувачів [10].

Щоб краще зрозуміти сам стандарт, розглянемо приклад веб-додатку, який допомагає користувачам планувати поїздки. Як частина функціональності він вміє аналізувати користувачів пошти на наявність листів з підтвердженнями бронювання і автоматично включити їх в плануваний маршрут. Виникає питання, як це веб-приложение може безпечно отримати доступ до поштових користувачів, наприклад, до Gmail. Існує два варианти:

- попросити користувача вказати дані свого облікового запису? — поганий варіант;
- попросити користувача створити ключ доступу? — можливо, але дуже складно.

Як цю проблему і дозволяє вирішити стандарт OAuth: він описує, як прикладне програмне забезпечення для подорожей (клієнт) може отримати доступ до поштового сервера (ресурс-сервер) з дозволу користувача (власника ресурсу). В загальному вигляді весь процес складається з декількох кроків (рисунок 1.8):

- користувач (власник ресурсу) дає дозвіл додатку (клієнту) на доступ до певного ресурсу у вигляді гранту.
- додаток звертається до сервера авторизації і отримує токен доступу до ресурсу в обмін на свій грант. У нашому прикладі сервер авторизації — Google. При виклику застосунок додатково аутентифікується за допомогою ключа доступу, що буд виданий йому при попередній реєстрації.
- додаток використовує цей токен для отримання потрібних даних від серверних ресурсів (в нашому випадку - Gmail сервіс).

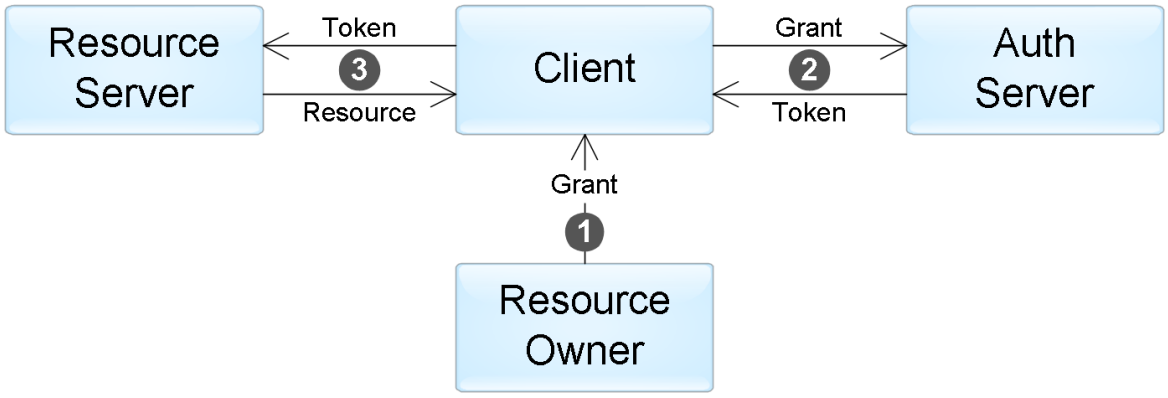

Рисунок 1.8 Схема взаємодії компонентів в стандартах OAuth.

Стандарт описує чотири види грантів, які визначають можливі сценарії застосування:

- Authorization Code — цей грант може отримати від сервера авторизації після успішної аутентифікації та підтвердження погодження на надання доступу. Така способ найчастіше використовується в веб-додатках. Процес отримання

гранту дуже подібний механізму аутентифікації пасивних клієнтів у SAML та WS-Federation;

- Implicit - застосовується, коли у додатку немає можливості безпечно отримати токен від сервера авторизації (наприклад, JavaScript-додаток у браузері). У цьому випадку грант представляє собою токен, отриманий від сервера авторизації, а крок № 2 виключається зі сценарію вище;
- Resource Owner Password Credentials — грант представляє пару логін / пароль користувача. Может застосовуватися, якщо приложение є «інтерфейсом» для сервера ресурсів (наприклад, приложение - мобільний клієнт для Gmail).
- Client Credentials — у цьому випадку немає ніякого користувача, а додаток отримує доступ до своїх ресурсів за допомогою своїх ключів доступу (виключений крок № 1).

Стандарт не визначає формат токена, який отримує додаток: у сценаріях, адресних стандартом, додатку немає необхідності аналізувати токен, т. к. він тільки використовується для отримання доступу до ресурсів. Тому ні токен, ні грант самі по собі не можуть бути використані для автентифікації користувача. Однак, якщо прикладну програму необхідно отримати достовірну інформацію про користувача, існують кілька способів зробити це:

- зазвичай API сервера ресурсів включає операцію, що надає інформацію про самого користувача (наприклад, / me в Facebook API). Приставка може виконати цю операцію кожен раз після отримання токена для ідентифікації клієнта. Така методика іноді називають псевдо-аутентифікацією;
- використовуйте стандарт OpenID Connect, розроблений як слой облікових даних на поверх OAuth (опубліковано в 2014 році). Відповідно до цього стандарту, сервер авторизації надає додатковий токен ідентичності на кроку № 2. Цей токен у форматі JWT буде містити набір визначених полів (претензій) з інформацією про користувача.

Варто відзначити, що OpenID Connect, замінивши попередні версії стандарту OpenID 1.0 і 2.0, також містить набір необов'язкових доповнень для пошуку

серверів авторизації, динамічної реєстрації клієнтів та управління користувачами сесії.

Висновки до розділу 1

У **першому розділі** було:

1. Проаналізовано сучасні методи авторизації користувачів. Були розглянуті усі види аутентифікації користувачів та стандарти авторизації та були визначені переваги та недоліки.

2. Аналіз досліджень показав, що системи хмарних обчислень використовують аутентифікації по ключ-доступу, а даний вид аутентифікації неможливо використовувати в системах з мікросервісною архітектурою.

3. Аналіз досліджень показав, що необхідно розробити фреймворк, що буде вирішувати дану проблему.

2. АВТОРИЗАЦІЯ КОРИСТУВАЧІВ У СИСТЕМАХ ХМАРНИХ ОБЧИСЛЕНЬ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ

Архітектурний стиль мікросервісов — це підхід, при якому єдине додаток будується як набір невеликих сервісів, кожен з яких працює у власному процесі і взаємодіє з іншими використовуючи легковажні механізми, Як правило, взаємодія організована на основі протоколу передачі даних HTTP. Ці сервіси побудовані навколо бізнес-потреб і розгортаються незалежно, з використанням повністю автоматизованої середовища. Існує абсолютний мінімум централізованого управління цими сервісами. Самі по собі ці сервіси можуть бути написані на різних мовах і використовувати різні технології зберігання даних. Зазвичай, взаємодія між мікросервісами відбувається у вигляді REST запитів, що і робить кожен сервіс незалежним один від одного у виборі технологій.

Аутентифікація користувачів у системах з мікросервісною архітектурою відбувається лише на основі токенів, адже незалежність сервісів один від одного робить неможливим використання сесій, а призводить до неможливості використання жодного з наведених методів описаних вище. Отже для авторизації можливе використання єдиного способу аутентифікації, а оскільки системи хмарних обчислень використовують у якості методу аутентифікації ключ-доступ, це викликає потребу у реалізації додаткового шару, що з’єднає ці два способи в один. Цю проблему і вирішує розроблений фреймворк.

2.1. Авторизацію користувачів у мікросервісній архітектурі

У традиційній багаторівневій архітектурі застосунку(рисунок 2.1), веб-інтерфейс на стороні сервера стосується автентифікації користувача, викликаючи реляційну базу даних або сервер LDAP. Потім створюється HTTP-сеанс, що містить

необхідну автентифікацію та дані користувача. Контекст безпеки поширюється між рівнями на сервері додатків, тому немає потреби повторно автентифікувати користувача.

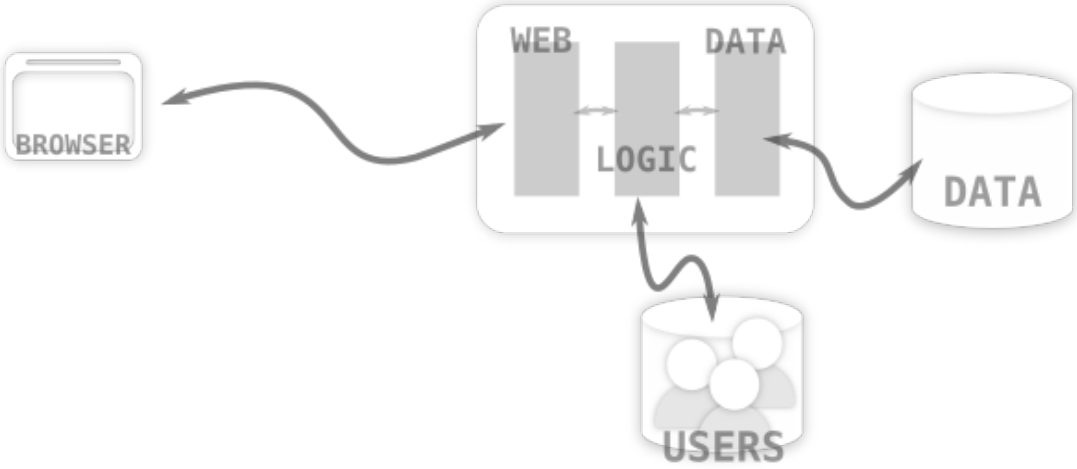

Рисунок 2.1 Багаторівнева архітектура застосунку.

За допомогою архітектури мікросервісу відсутня веб-програма на стороні сервера, замість цього ми маємо HTML5 і мобільні додатки на стороні клієнта. Програми запускають кілька служб, які, у свою чергу, можуть викликати інші служби. У цій архітектурі вже не існує жодного шару, який може мати справу з аутентифікацією, і, як правило, це stateless застосунок, тому HTTP-сеансу немає (рисунок 2.2).

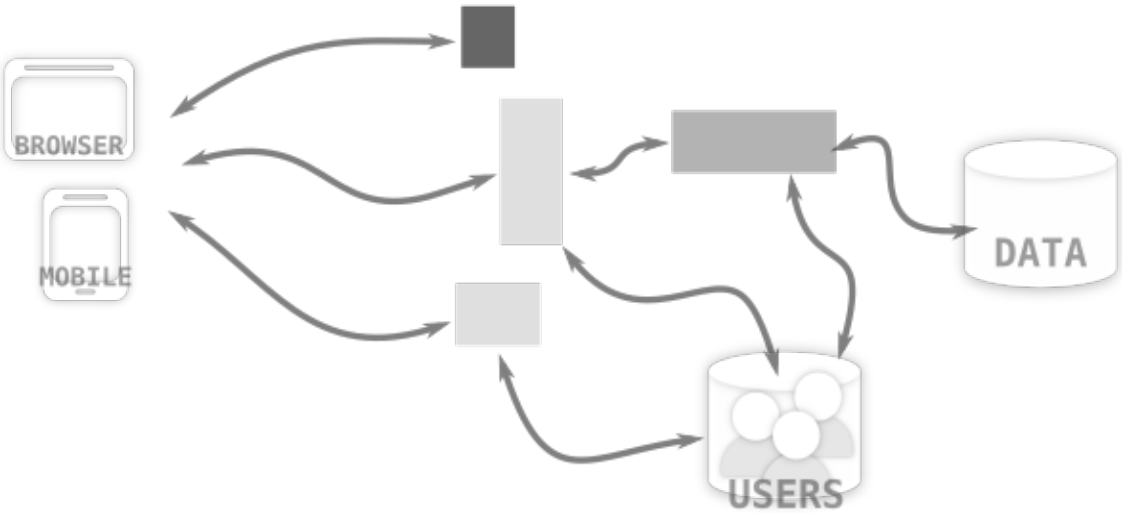

Рисунок 2.2 Мікросервісна архітектура застосунку.

Оскільки мікросервіси — це набір малих сервісів, кожен з яких вирішує одне конкретне завдання, очевидним рішенням безпеки є служба аутентифікації та авторизації. Саме тут Keycloak та OpenID Connect приходять на допомогу. Keycloak надає послуги, необхідні для авторизації [11].

Першим кроком до забезпечення безпеки мікросервісів є аутентифікація користувача. Це робиться шляхом додав

ання адаптера Java Keycloak у наш додаток. Для мобільних додатків є адаптер Keycloak Cordova, але існує також нативна підтримка завдяки проекту AeroGear.

У додатку потрібно просто додати кнопку входу. Коли користувач натискає кнопку входу, браузер користувача перенаправляється на екран входу на сервері Keycloak. Після цього користувач аутентифікує з сервером Keycloak. Оскільки аутентифікація виконується сервером Keycloak, а не вашою програмою, легко додати підтримку багатофакторної аутентифікації або логінів через соціальні мережі без необхідності будь-яких змін у вашій заявці (рисунок 2.3) [12].

Рисунок 2.3 Використання KeyCloak для авторизації.

При авторизації користувача Keycloak надає програмі токен має (рисунок 2.4). Токен містить інформацію про користувача, а також дозволи, які він має.

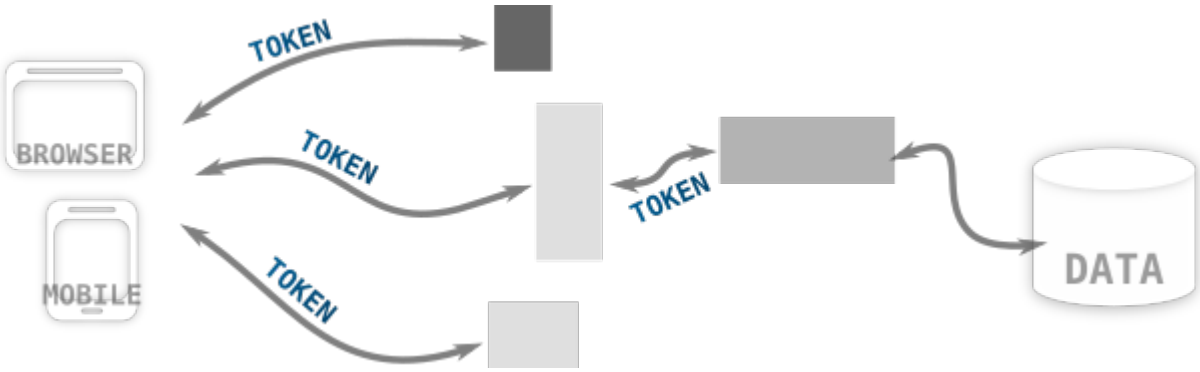

Рисунок 2.4 Використання KeyCloak для авторизації.

Далі необхідно захистити мікросервіси. Знову ж таки, з Keycloak має адаптери, для легкого налаштування: JavaEE, NodeJS та інші. Якщо у нас немає адаптера, то досить легко перевірити токени самостійно. Токен в основному є лише підписаним документом JSON, і його можна перевірити безпосередньо самими сервісами або за допомогою сервера Keycloak.

Якщо сервіс повинен викликати інший, то він можепросто передати отриманий токен, який буде викликати інший сервіс зтими самими правами користувача.

2.2. Архітектура систем хмарних обчислень

Хмарні обчислення ґрунтуються на об'єднанні старих і нових концепцій в різних областяхдосліджень, таких як сервіс-орієнтована архітектури, розподілені і мережеві обчислення, віртуалізація [13]. Cloud Computing можна вважати новою парадигмою обчислень, яка дозволяє користувачам використовувати тимчасову обчислювальну інфраструктуру, надаємо ую як сервіс провайдером хмари з можливістю одного або декількох рівнів абстракції. На поточний момент кожен хмарний сервіс має свій відрізняється інтерфейс взаємодії з користувачем. При цьому не існує єдиного інтерфейсу для інтегрованого доступу до послуг хмарних обчислень, і необхідна розробка шлюзів для забезпечення такої можливості. Розподілені комп'ютерні системи Cloud Computing надають послуги з надання віртуальнихсередовищ, при цьому необхідно контролювати інтерфейси управління віртуальними машинами, пам'яттю, системами зберігання, а також забезпечити необхідну пропускну здатність і прозоре управліннявеликомасштабної інфраструктурою, що складається з тисяч системних компонентів [14].

Технологія Grid використовується для створення розподіленої обчислювальної інфраструктури та забезпечує інтеграцію інформаційних і обчислювальних ресурсів на основі мережевих технологій і спеціального програмного забезпечення проміжного рівня, а також набору стандартизованих служб для забезпечення надійного колективного доступу до розподілених ресурсів.

Основною відмінністю Cloud інфраструктури від Grid обчислень, є широкомасштабне розгортання технологій віртуалізації. Таким чином, хмара містить додатковий шар віртуалізації, який є середовищем виконання і хостингу прикладних сервісів. Cloud виртуализирует всі фізичні ресурси, а також інфраструктуру, в вигляді налаштованої і готової для використання віртуальної машини [15].

На рисунку 2.5 показана багатошарова архітектура Cloud Computing, яка складається з наступних рівнів: рівень користувача, рівень Cloud, рівень GRID.

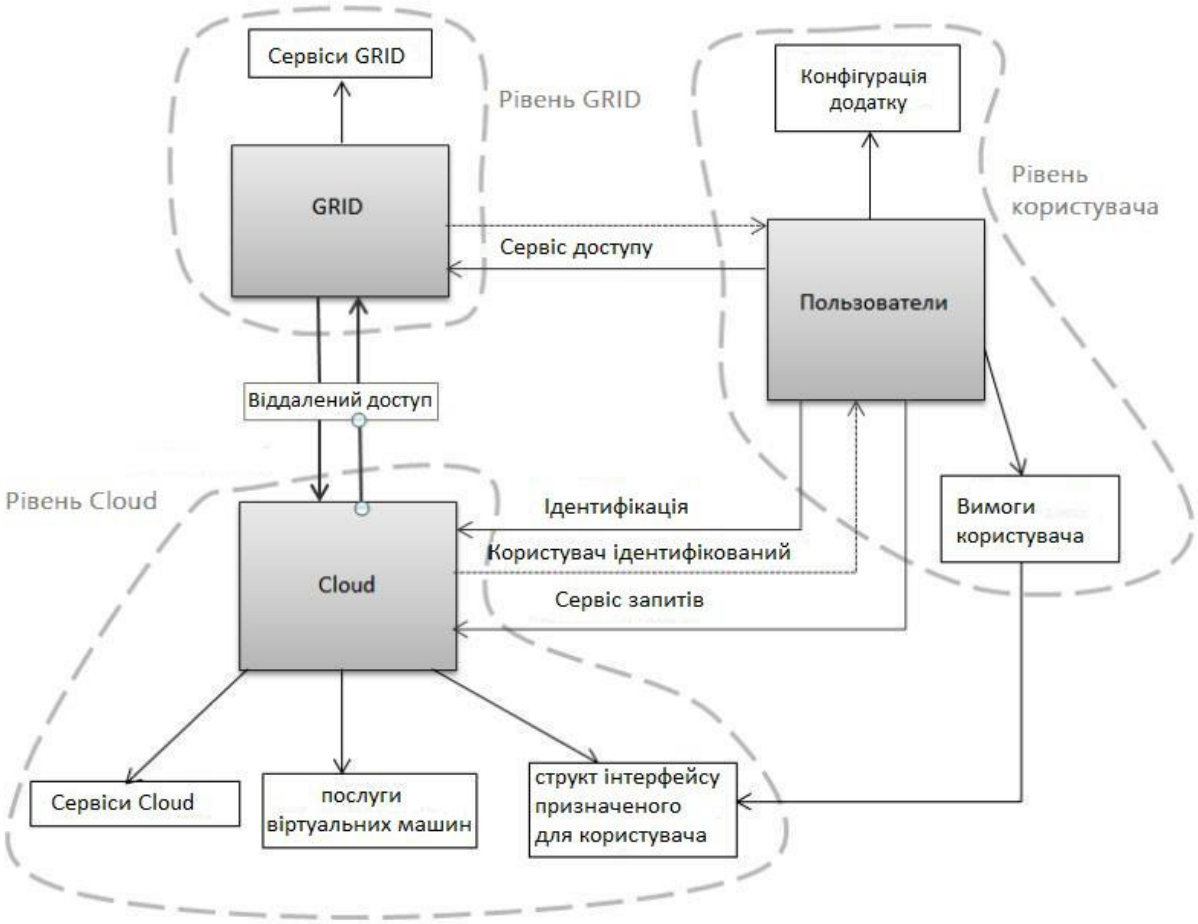

Рисунок 2.5 Багатошарова архітектура систем хмарних обчислень.

Адміністратор Cloud управляє інфраструктурою і відповідає за обслуговування запитів користувачів згідно специфікації. Після отримання запиту користувача адміністратор забезпечує віртуальну машину зазначеної конфігурації, запускає її на одному з комп'ютерів Cloud інфраструктури та надає доступ користувачеві.

Наведемо процедура створення віртуальної машини за запитом користувача. На першому етапі виконується реєстрація користувача, потім користувач входить в систему і подає запит адміністратору з необхідними характеристиками віртуальних машин (операційна система, пам'ять, жорсткий диск, процесор, програмне забезпечення). Адміністратор створює шаблон для віртуальної машини, виконує пошук відповідного кластера для розміщення віртуальної машини необхідної конфігурації. Виконується копіювання образу операційної системи з шаблону, хід розгортання віртуальної машини контролює гипервизор.

Віртуальні машини розташовані на кластерах, користувачеві надається доступ до віртуальної машині за допомогою технології віддаленого доступу до робочого столу. Даний підхід має наступну специфіку:

- мобільність — інфраструктура знаходиться в хмарі, а користувачі отримують доступ через мобільні пристрою з будь-якого місця;
- масштабованість: хмара надає обчислювальну середу в найкоротші терміни, і користувачеві не потрібно турбуватися про брак обчислювальних ресурсів;
- гнучкість: користувач може розширювати можливості інфраструктури, і гнучко змінювати обчислювальну платформу.

У проведеному аналізі різних типів авторизації було виявлено, що хмарні системи використовують авторизації за ключем доступу, а отже спираючись на знання архітектури систем хмарних обчислень та підхід до авторизації в системах з мікросервісною архітектурою, можна розробити фреймворк, що вирішує проблему авторизації.

2.3. Фреймворк для авторизації в системах хмарних обчислень з мікросервісною архітектурою

Спираючись на проведений аналіз систем авторизації, було вирішено розробити фреймворк, що вирішить проблему використання мікросервісної архітектури для систем хмарних обчислень шляхом обертання усіх запитів до

сервера, тобто запити будуть надходити спочатку в спеціальний сервіс, що буде перевіряти ключ доступу користувача, а потім відправляти в сервіс авторизації мікросервісів (для розробки було обрано сервіс авторизації KeyCloak). На рисунку 2.6 зображено схему роботи фреймворку.

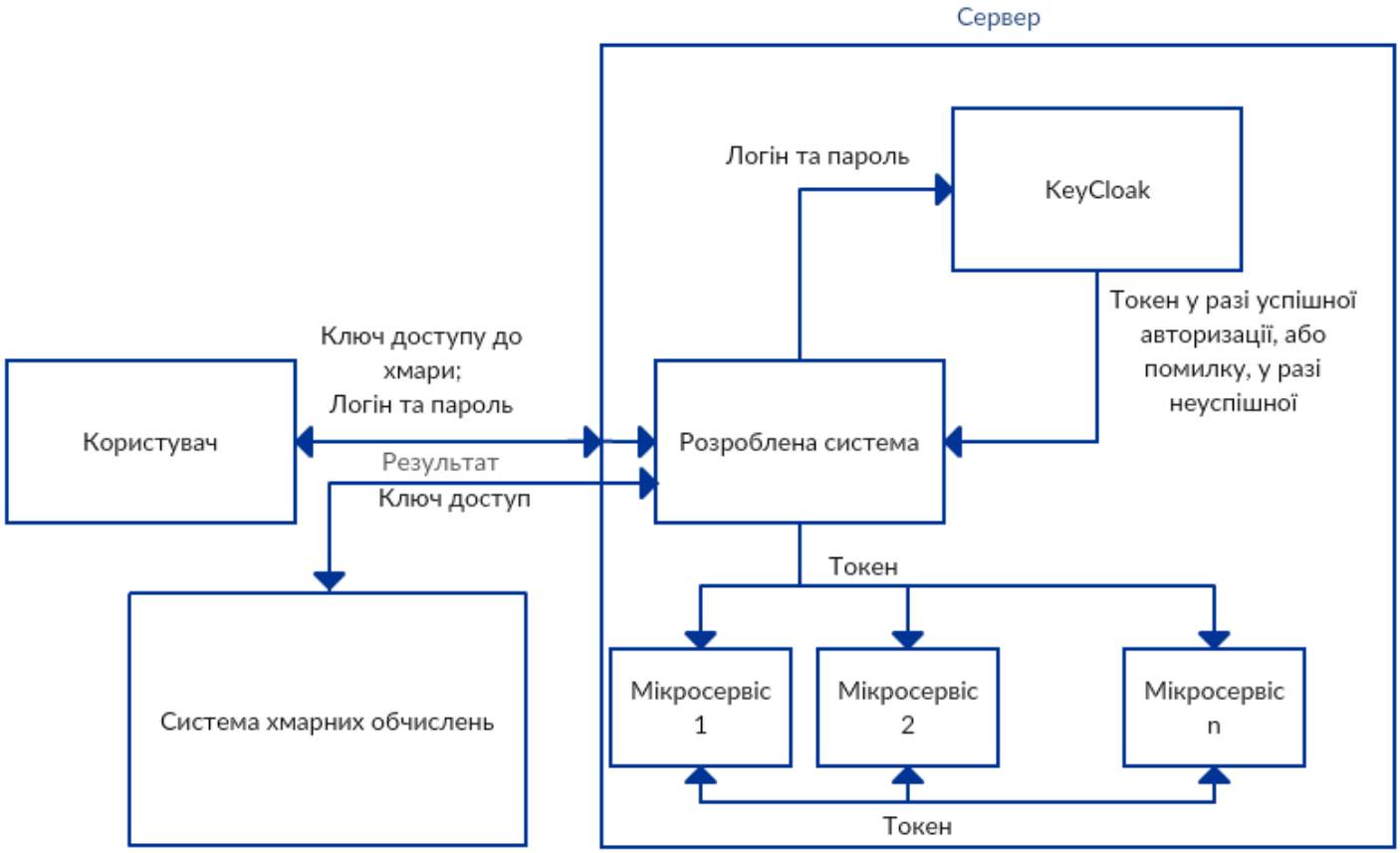

Рисунок 2.6 Схема роботи запропонованого рішення.

Як видно зі схеми, користувач відправляє на сервер всій логін, пароль та ключ-доступ, далі розроблений сервіс перехоплює запит та відправляє його у сервіс авторизації KeyCloak, де перевіряється логін та пароль, а також рівень доступу користувача. На основі цих даних у розроблений сервіс повертається відповідь про те, чи успішно пройшой процес аутентифікації користувача, у разі неуспішності відправляється одразу відповідь користувачу про невдалу спробу авторизуватися; у разі ж, успішної аутентифікації, розроблений сервіс перенаправляє запит у необхідний мікросервіс. Далі мікросервіси взаємодіють між собою і у разі необхідності взаємодії з системою хмарних обчислень відправляють запит до розробленої системи, яка у свою чергу, по токену, дістає ключ-доступ даного

користувача і віправляє запит до системи хмарних обичслень. Після підрахунків у хмарі, повертаєме значення знову транзитом через розроблену систему повертається до мікросервісу, що викликав дану функцію системи хмарних обчислень.

Висновки до розділу 2

У **другому розділі** було:

1. Проаналізовано сучасні методи авторизації у мікросервісній архітектурі.
2. Проаналізована архітектура систем хмарних обчислень.
3. Розроблено фреймворк вирішення проблеми авторизації у системах хмарних обчилсень з мікросервісною архітектурою.

3. РОЗРОБКА ФРЕЙМВОРКУ ДЛЯ АВТОРИЗАЦІЯ КОРИСТУВАЧІВ У СИСТЕМАХ ХМАРНИХ ОБЧИСЛЕНЬ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ

Одним із найважливіших завдань при розробці програмних продуктів є вибір таких засобів, які б полегшили роботу програміста, надавши всі необхідні інструменти для реалізації поставленого завдання, і дали б змогу отримати результат, який повністю задовольняє користувача.

При створенні програмного продукту було використано засоби реалізації, що зображені на рисунку 3.1.

Рисунок 3.1 — Засоби реалізації програмного забезпечення

Як зображено на рисунку 3.1, при створенні програмного забезпечення були

використані такі засоби реалізації:

- середовище розробки IntelliJ IDEA, адже має найбільше функціоналу серед конкурентів;
- мову програмування Java для написання серверної частини;
- фреймворк Spring, а саме його модулі — Core Container, Web, Data Access/Integration та Boot, для організації архітектури та взаємодії серверної частини з інтерфейсом та базою даних;
- технологію Maven для підключення всіх модулів та розгортання застосунку у середовищі хмарних обчислень";
- Git для версіювання розробленої системи;
- базу даних PostgreSQL, адже вона підтримує розподіленність;
- Lombok для збільшення чистоти коду;
- Amazon Web Services було використано для організації хмарної системи;
- Docker для створення необхідного оточнення для AWS;
- Hibernate для простішого з'єднання коду з базою даних;
- KeyCloak, як сервер аутертифікації.

Вибір технологій зумовлений тим, що застосунок повинен не залежити від пристроя з якого буде виконуватися вхід у систему, а також від пристроя на якому буде розгортатися застосунок, саме тому були обрані незалежні від операційної системи технології.

Мовою програмування високого рівня була обрана Java, адже вона окрім, вже зазначеної кроссплатформеності, має багато бібліотек, що допомагають спростити процес налаштування застосунку, а також стандартизують архітектуру та стиль написання коду, що важливо для можливого розширення функціоналу іншими розробниками.

Базою даних була обрана PostgreSQL в першу чергу через швидкодію, адже при накопиченні даних це відіграє важливу роль у продуктивності роботи. Для об'єктно-реляційного відображення було використано Hibernate, а для автоматичної генерації запитів до бази даних — Spring Data JPA.

У якості системи хмарних обчислень було використано Amazon Web Services,

адже дана система має великий арсенал функцій та налаштувань, а також порівняно з іншими системами коштує не багато.

3.1. Мова програмування Java

Уся логіка процесів системи написана мовою високого рівня Java — об’єктно-орієнтованою мовою програмування. Вибір зумовлений тим, що платформа Java має велику кількість технологій та фреймворків, що були створені для вирішення широкого спектру прикладних задач. Іншим достоїнством Java є віртуальна машина, що забезпечує кросплатформеність.

Окрім кросплатформеності сильною стороною цієї мови є також висока надійність роботи, адже вона розроблялася як об’єктно-орієнтована мова високого рівня з жосткою типізацію. Так, для запобігання неоднозначності та спрощення розуміння коду з мови було виключено множинне наслідування. Замість цього, було введено поняття інтерфейсу, що являє собою, так званий контракт для класів, що будуть наслідуватися від нього. Іншим чинником високої надійності цієї мови програмування є система винятків та обробки помилок, які були поділенні на два види:

- виняткові ситуації, що можуть виникнути під час роботи програми, які обов’язково розробник має обробити, наприклад: користувач увів у поле у якому допустимі символи, або програма намагається відкрити файл на системі, якого не існує;
- ситуації, коли програма зустрічається з неочікуваними труднощами, наприклад: операція над елементами масиву поза його межами, або переповнення пам’яті через допущену помилку програмістом [16].

Для запобігання останнім ситуаціям, в обраній мові програмування є вбудований сервіс керування пам’яттю, що звільняє з оперативної пам’яті комп’ютера дані, які не використовуються. Таким чином, програміст вирішує, коли створювати об’єкт, а віртуальна машина відповідає за звільнення пам’яті після того, як об’єкт стає непотрібним — немає посилань на нього, збирач сміття

автоматично прибирає цей об’єкт з пам’яті. Java не підтримує вказівники, як мови C та C++, наприклад. Це зроблено для того, щоб дозволити збирачу сміття переміщувати вказівники об’єкту з однієї області пам’яті в іншу.

Об’єктно-орієнтований підхід до написання коду дозволяє оперувати поняттями, що зустрічаються в реальному житті з певною долею абстракії [17]. Парадигма ООП наділила мову такими властивостями як масштабованість, що дає змогу неодноразово розширювати розроблену систему. Розширюваність системи полягає в тому, що в систему можна додавати нові компоненти, без зміни вже існуючих. Іншою перевагою цього підходу є багаторазове використання написаного коду, що значно скорочує кількість написаного коду.

Наразі Java являє собою одну з найрозповсюдженіших мов програмування, адже може застосовуватися для написання багатьох типів програм.

3.2. Фреймворк Spring

Для створення архітектури систем з використання веб-інтерфейсу використовують сучасні бібліотеки, чи фреймворки, які спрощують процес написання коду.

Програмний фреймворк (англ. software framework) — це комплект готових до використання комплекс програмних рішень, що включають в себе архітектуру побудови проекту, логіку та базову функціональність системи або підсистеми та, навіть, дизайн.

Фреймворк містить в собі набір бібліотек, що пропонують готовий набір рішень, також може містити допоміжні програми, скрипти та загалом все те, що полегшує створення та поєднання різних компонентів великого програмного забезпечення чи швидке створення готового і не обов’язково об’ємного програмного продукту. Одна з головних переваг використання фреймворків — це стандартизованість структури застосунку [18].

Spring — це фреймворк з відкритим вихідним кодом, що вільно розповсюджується, створений з метою спрощення розробки корпоративних

додатків, забезпечує комплексну модель розробки і конфігурації для сучасних бізнес-додатків на Java. Ключовий елемент Spring — це підтримка інфраструктури на рівні програми: основна увага приділяється "водопроводу" бізнес-додатків, тому розробники можуть зосередитися на бізнес-логіці без зайвих налаштувань в залежності від середовища виконання [19]. Spring можна використовувати для побудови будь-якої програми на мові Java (тобто автономних, веб додатків, додатків JEE і т.д.), що позитивно відрізняє Spring від багатьох інших платформ [20].

Для спрощення розробки додатків мовою Java у основі Spring лежать наступні стратегії:

- слабке зв’язування шляхом ін’єкції залежностей через конструктор, або сетери (англ. setter) та організація взаємодії через інтерфейси, цей підхід дозволяє легше тестувати застосунок, а також дозволяє координувати роботу кожного об’єкта в системі від третьої сторони;
- інверсія управління, що дозволяє писати незалежні один від одного компоненти, що надає переваги при розробці застосунку в команді, перенесенні та заміні модулей;
- декларативне програмування через аспекти та загальноприйняті угоди, що полегшує сприйняття коду;
- використання простих Java об’єктів, або їх ще називають POJO (Plain Old Java Object), без зобов’язання реалізовувати, чи наслідувати специфічні для фреймворку інтерфейси та класи відповідно;
- Spring виступає у ролі контейнера для об’єків, а отже, управляє їх життєвим циклом.

Ін’єкція залежностей (англ. Dependency Injection) — це паттерн проектування та архітектурна модель, що описує ситуацію, коли один об’єкт реалізує свій функціонал через інший об’єкт. Об’єкт передає керування створенням необхідних йому залежностей зовнішньому, спеціально призначеному для цього механізму. Наприклад, з’єднання з базою даних передається конструктору об’єкта через аргумент, замість того, щоб контруктор сам встановлював з’єднання. Перевагами

цього підходу є те, що він дозволяє відділити об’єкти від реалізації механізмів, котрі він використовує, в наслідок чого, отримаємо гнучкість в розробці застосунку.

Spring складається з модулів (рисунок 3.2), що незалежні один від одного, окрім модуля Core Container, який являється ядром і до нього приєднують всі інші модулі. Цей модуль з чотирьох складових: Beans, Core, Context, SpEL [21].

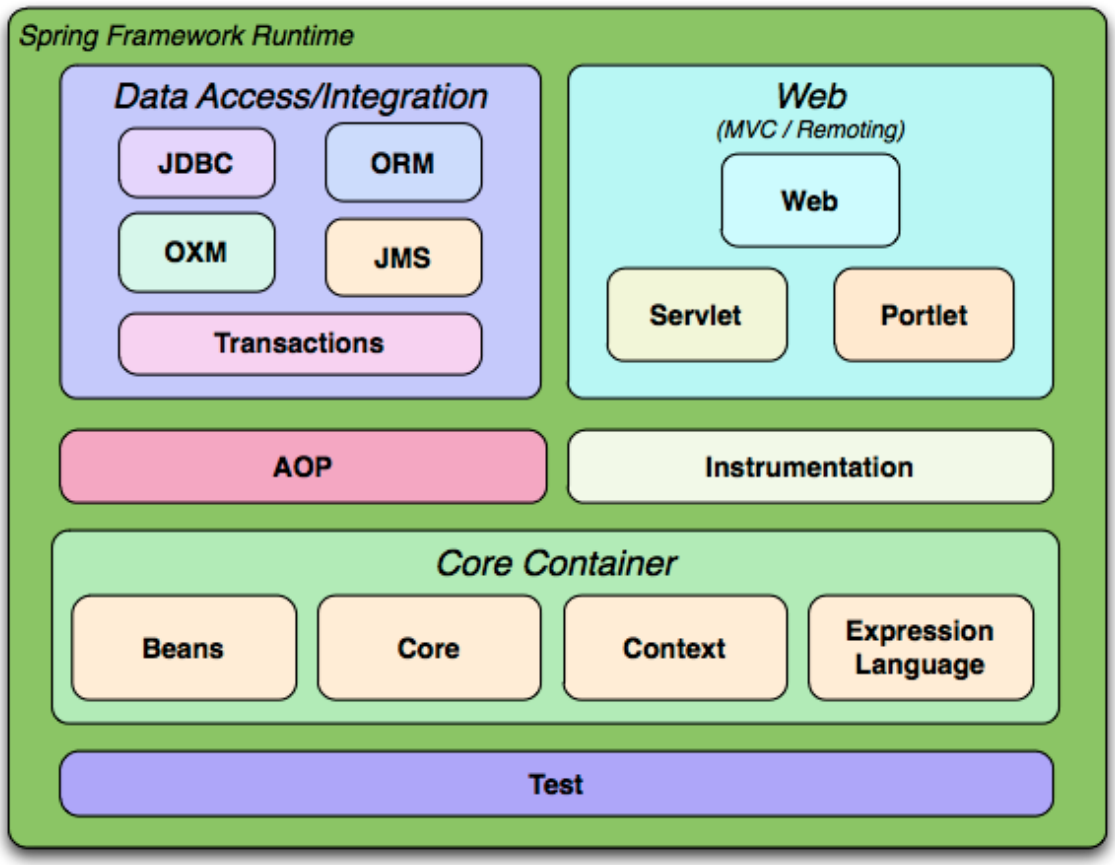

Рисунок 3.2 — Модулі Spring фреймворку

Beans та Core забезпечують найбільш фундаментальні частини фреймворку — ін’єкцію залежностей та інверсію управління, забезпечую реалізацію шаблона фабрики, що усуває необхідність реалізовувати паттерн Singleton та відокремлює конфігурацію з специфікацією від фактичної логічки програми. Context надає можливість отримувати доступ до об’єктів та додає підтримку інтернаціоналізації. SpEL (англ. Spring Expression Language) додає підтримку мов запитів, а саме: встановлення і отримання значень властивостей, зчитування з конфігураційного файлу, виклик методу, доступ до контексту масивів, колекцій та індексаторів,

логічних і арифметичних операцій, пошук об'єктів по імені.

Модуль Data Access/Integration складається з компонентів, що відповідають за роботу з базою даних. Так, підмодуль ORM (англ. Object-relational mapping) надає шар програмної реалізації взаємодії з популярними бібліотеками для вирішення задач об’єктно-реляційного відображення. Наприклад, з: JPA, JDO, Hibernate та iBatus. Підмодуль JDBC (Java DataBase Connectivity), що створює шар програмної реалізації однойменного стандарту взаємодії Java застосунку з різними СУБД. Підмодуль Transaction, як зрозуміло з назви — за підтримку та обробку транзакцій.

Модуль AOP додає підтримку аспектно-орієнтованого програмування в застосунок, що надає можливість ще гнучкіше налаштовувати його. Модуль Instrumentation надає можливість контролювати продуктивність застосунку. Модуль Test містить в собі бібліотеки для написання модульних та інтеграційних тестів.

Модуль Web забезпечує основний функціонал веб-застосунків. Компонент Web-Servlet надає реалізацію шаблону проектування архітектури проекту MVC (англ. Model View Controller — модель-представлення-контроллер) та HTTP протоколу для веб-застосунку. На рисунку 3.3 зображено архітектуру цього шаблону. За цим шаблоном забезпечується чітке розділення доменної (англ. domain) моделі, у якій міститься бізнес логіку від веб форм. Web Socket – модуль, що надає підтримку двошляхової комунікації між клієнтом та сервером.

MVC ви можете використовувати будь-який об'єкт в якості команди або об'єкта зі зворотним зв'язком; вам немає необхідності реалізовувати будь-якої спеціальний інтерфейс фреймворка або базовий клас. Завдяки гнучкості зв'язування даних в Spring, невідповідність типів розглядається як помилки валідації і тому це може бути оброблено в додатку, а не в якості системних помилок. Таким чином, вам не потрібно дублювати властивості бізнес-об'єктів, як простих нетипізований рядків для ваших об'єктів форм. Тому можна легко обробляти неправильні підтвердження або правильно конвертувати їх в рядки. Замість цього, бажано пов'язувати такі об'єкти безпосередньо з об'єктами бізнес логіки.

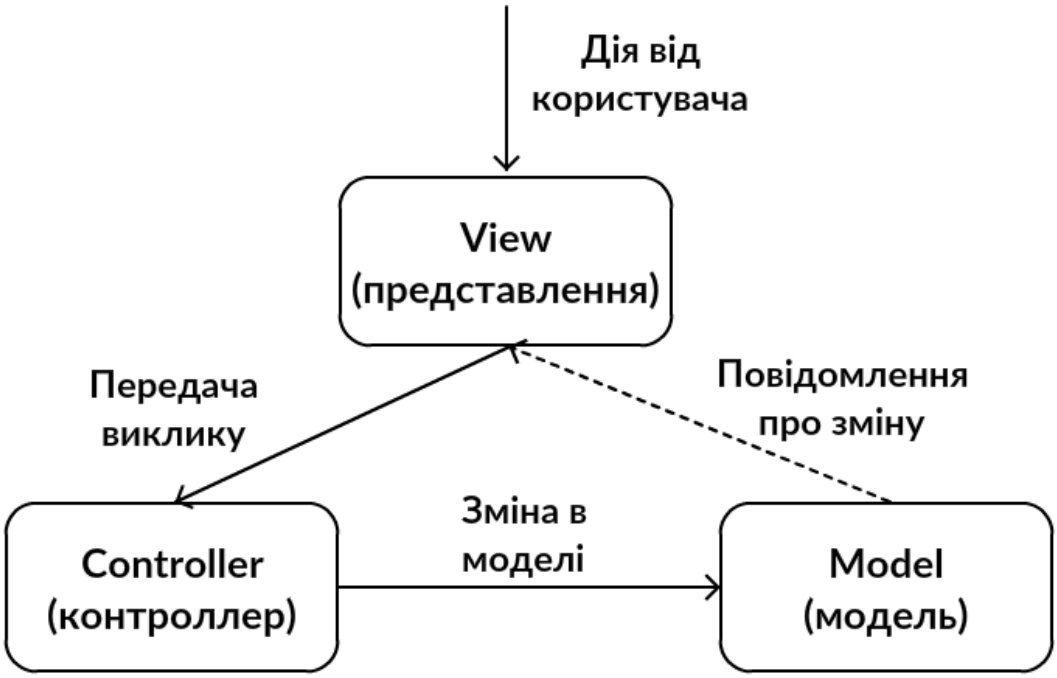

Рисунок 3.3 — Шаблон проектування MVC

Цей модуль Spring побудований навколо центрального сервлету, який називається Dispatcher Servlet, котрий розподілює запити між контролерами. В ньому налаштовується обробка запитів, локалізація та часові зони [22]. Dsipatcher Servlet (з англ. Диспетчер сервлетів), це звичайний сервлет, що реалізує протокол HTTP. На рисунку 3.4 показана схема роботи цього сервлету з програмою. Спочатку запит потрапляє до Dispatcher Servlet, після чого він передає його у Handler Mapping, що переглядає усі наявні контролери та знаходить той, який знає як обробляти цей запит та повертає його ім’я назад до Dispatcher Servlet.

У ситуації коли буде знайдено два контролера, що мають необхідний функціонал обробки запиту неможлива, адже, це на етапі запуску проекту програмісту буде видана помилка. Після отримання імені контролера запит передається у нього. В контролері відбувається обробка запиту та назад відправляється об’єкт ModelAndView, що містить в собі дані та як їх потрібно відображати. Dispatcher Servlet на основі цього об’єкту шукає відповідне представлення (ім’я об’єкт типу View) за допомогою View Resolver. Завдяки цьому диспетчер сервлетів знає у яке представлення відправити модель (об’єкт типу Model), що містить дані. Від модуля представлення диспетчер сервлетів отримує

відповідь, яка потім відсилається на веб-частину застосунку, у випадку коли вона необхідна.

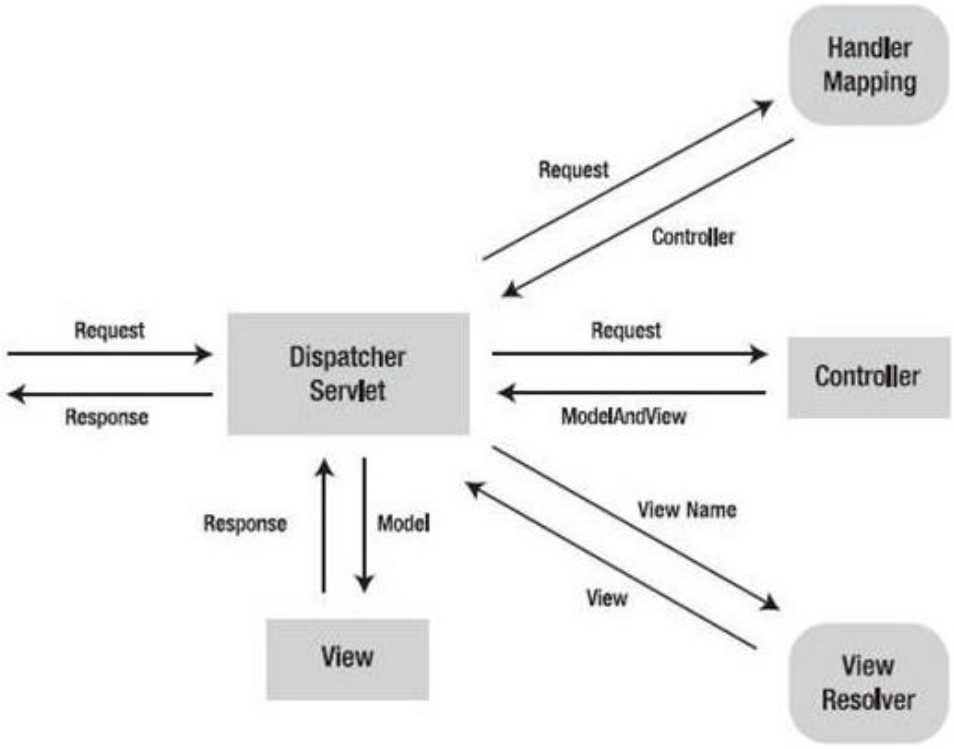

Рисунок 3.4 — Схема роботи Dispatcher Servlet

Spring організовує архітектуру проекту шляхом зобов’язування програміста вказувати через анотації до якого типу класів відноситься даний, якщо ж не вказати тип, то фреймворк не буду поміщати даний клас у свій контейнер, а отже в ньому не будуть працювати методи з бібліотек, що надає Spring. Існує 4 анотації, що утворюють архітектуру проекту:

- @Component позначається приналежність класу до фреймворку;
- @Controller позначається клас, в якому Dispatcher Servlet буде шукати метод, що оброблює запит;
- @Service позначається клас, в якому реалізується бізнес логіка, нічим кардинально не відрізняється від @Component, але дозволяє розробнику вказати смислове навантаження класу;
- @Repository позначається клас, що буде використовуватися для роботи з пошуком, отриманням та зберіганням даних. Здебільшого дані класи

використовуються для організації взаємодії з базою даних та реалізації відповідних шаблонів.

У результаті використання даного фреймворку отримаємо архітектуру проекту як показано на рисунку 3.5.

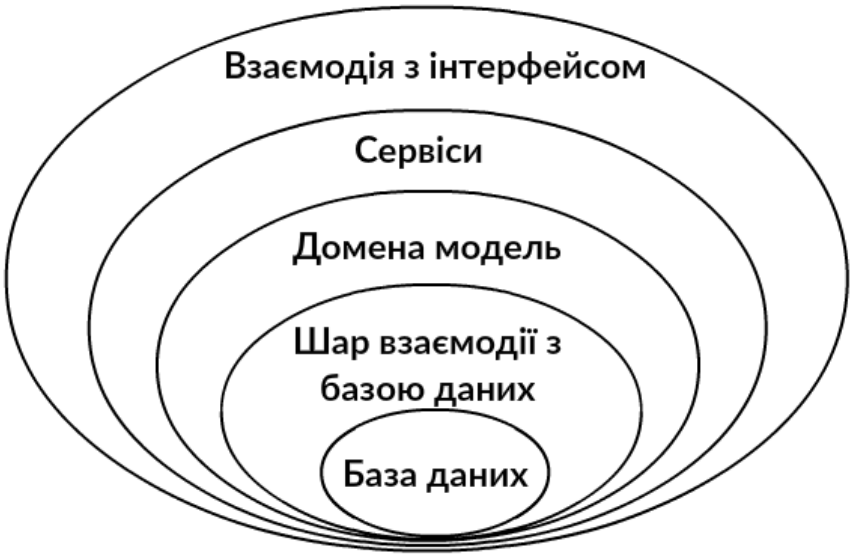

Рисунок 3.5 — Архітектура проекту з використання Spring фреймворку

3.3. Фреймворк Spring Data JPA

Фреймворк Spring Data JPA реалізує шар доступу до даних, який може бути досить громіздким. Занадто багато шаблонного коду пишеться для реалізації таких завдань, як розбивка на сторінки і аудит. Spring Data JPA покликане значно поліпшити реалізацію шару доступу до даних, скоротивши зусилля на те, що дійсно необхідно. Як розробник, ви пишете інтерфейс сховища, включаючи власні методи пошуку, а Spring забезпечує їх автоматичну реалізацію [23]. Spring Data JPA надає ряд інтерфейсів, в яких уже описуються найбільш часто використовувані методи роботи з базою даних. Роздивимося на прикладі одного з основних інтерфейсів — JPA Repository (рисунок 3.6).

Наслідуючи цей інтерфейс програміст отримує готову реалізацію методів збереження , оновлення та видалення даних з бази, а також методи пошуку всіх елементів з талички та за ід. Для того, щоб власний створений метод у інтерфейсі,

що наслідую інтерфейс Spring Data JPA, виконував запити в базу, потрібно щоб назва методу будувалася зі слів, які зрозумілі декомпілятору даного фреймворку, або напряму через анотацію над методом вказувати SQL запит.

```
@NoRepositoryBean
public interface JpaRepository<T, ID extends Serializable> extends PagingAndSortingRepository<T, ID>
    List<T> findAll();

    List<T> findAll(Sort var1);

    List<T> findAll(Iterable<ID> var1);

    <S extends T> List<S> save(Iterable<S> var1);

    void flush();

    <S extends T> S saveAndFlush(S var1);

    void deleteInBatch(Iterable<T> var1);

    void deleteAllInBatch();

    T getOne(ID var1);

    <S extends T> List<S> findAll(Example<S> var1);

    <S extends T> List<S> findAll(Example<S> var1, Sort var2);
}
```

Рисунок 3.6 — Інтерфейс Spring Data JPA

Наприклад, метод з іменем: findAllByField(FieldType fieldValue) виконає запит до бази аналогічний до: “SELECE * FROM TABLE WHERE field = fieldValue” [24].

3.4. Фреймворк Spring Boot

Spring Boot дозволяє вам легко створювати повноцінні, виробничого класу Spring-додатки, про які можна сказати - "просто запусти". Розробники включили Spring-платформу і сторонні бібліотеки, щоб ви могли запустити застосунок з мінімальними зусиллями. Більшості Spring Boot додатків потрібно зовсім маленька Spring-конфігурація. Можливості, що надає Spring Boot:

- створення повноцінних Spring додатків;
- вбудований Tomcat або Jetty (не потрібно установки WAR файлів);
- забезпечує 'початкові' POMs для спрощення вашої Maven конфігурації;
- автоматична конфігурірація Spring коли це можливо;

- забезпечує такими можливостями, як метрики, моніторинг станами і розширена конфігурація;
- абсолютно без генерації коду і без написання XML конфігурація;
- довідник містить докладний опис усіх можливостей, з додаванням коротких інструкцій найбільш загальних випадків використання.

Spring Boot включає в себе інструменти командного рядка, щоб ви могли швидко створити прототип Spring-додатки, наприклад. Це дозволяє вам запускати Groovy скрипти, які мають Java-подібний синтаксис, по суті - невеликі шаблони коду. Дотримуйтесь інструкцій в основний документації, якщо хочете встановити Spring Boot CLI. На рисунку 3.7 зображено мнімально неохідну Maven конфігурацію для запуску програмного продукту [25].

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.1.8.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.1.7.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Рисунок 3.7 — Конфігурація Spring Data Boot.

3.5. Фреймворк Hibernate

Hibernate є інструментом об'єктно-реляційного відображення для Java з відкритим вихідним кодом. Він надає фреймворк для відображення об'єктно-

орієнтованої моделі даних в традиційні реляційні бази даних [26]. Hibernate не тільки дбає про відображення Java класів в таблиці БД (і типів даних Java в типи даних SQL), але також забезпечує запит даних і пошукові засоби і може значно скоротити час розробки який витрачається на написання SQL і JDBC коду вручну.

Зіставлення Java-класів з таблицями БД здійснюється за допомогою конфігураційних XML файлів, або за допомогою Java анотацій. Забезпечуються можливості по організації відносини між класами один-до-багатьох та багато-до-багатьох. В доповення до управління асоціацію між об'єктами, Hibernate може також управляти рефлексивними відносинами, де об'єкт має зв'язок один-ко-многим з іншими екземплярами свого типу.

Цей фреймворк не зобов’язує реалізовувати ніякі інтерфейси для своєї роботи, вся приналежність до фреймворку організовується через анотації, що вказують Hibernate як їх використовувати при зіставленні з базою даних. Під час виконання програми він зчитує ці анотації і використовує цю інформацію для того, щоб добудувати запити для відправки в деяку реляційну БД.

Hibernate являється однією з реалізацій JPA (Java Persistence API) — специфікація API Java EE, котра надає можливість зберігати об’єкти. Так, згідно неї, сутності — POJO-класи зв’язуються з БД за допомогою анотації @Entity, або через xml файл. Такий клас має задовольняти деяким умовам:

- повинна мати пустий конструктор;
- не може бути вкладена, інтерфейсом чи enum (перерахуванням);
- не може бути з final (константою) та мати final поля;
- мати хоча б одне поле помічене анотацією @Id.
- Даний фреймворк складається з таких компонентів:
- EntityManagerFactory — фабрика EntityManager, що забезпечує екземпляри диспетчера об'єктів, всі екземпляри виконані з можливістю підключення до тієї же бази даних, щоб використовувати одні і ті ж параметри за замовчуванням, як це визначено в конкретної реалізації, і т.д. Ви можете підготувати кілька фабрик entity manager отримати доступ до кількох сховищ даних;

- EntityManager використовується для доступу до бази даних в конкретній одиниці роботи. Він використовується для створення та видалення стійких сутностей, щоб знайти об'єкти по їх первинному ключу ідентичності, і для запиту по всім організаціям;
- Persistence context являє собою набір екземплярів об'єкта, в якому для будь-якої постійної ідентичності об'єкта є унікальний екземпляр об'єкта. У persistence context, екземпляри сутностей і їх життєвий цикл управляється конкретним менеджером сутностей. Обсяг цього контексту може бути або транзакція, або розширена одиниця роботи [27].

3.6. Фреймворк Apache Maven

Apache Maven — це фреймворк для автоматизації збірки проектів на основі опису їх структури в файлах на мові POM (англ. Plain Object Model). Maven забезпечує декларативну, а не імперативну (на відміну від засобу автоматизації збирання Apache Ant) збірку проекту. У файлах опису проекту міститься його специфікація, а не окремі команди виконання. Всі завдання по обробці файлів, описані в специфікації, Maven виконує за допомогою їх обробки послідовністю вбудованих і зовнішніх плагінів. Використання maven зобов’язує використовувати стандартну структуру каталогів, що добре при спільній розробці. Так, у кореневому каталогу лежать тільки папки src, в якій знаходиться увесь код; папка target, в якій лежать всі створені під час роботи Maven файли та pom.xml файл у якому настроюється взаємодія Maven з проектом. В папці src лежать ще дві папки — main, в якій лежить реалізація програмного продукту та test — в якому лежать модульні та інтеграційні тести.

Maven керує життєвим циклом проекту. Він має набір фаз, що визначають порядок дій при його побудові [28]. Maven містить три незалежних порядка виконання:

- clean — життєвий цикл для очищення проекту;
- default — основний життєвий цикл, що включає в себе такі фази як:

- validate, що виконує перевірку, чи є структура повною та правильною;
- compile, що компілює код програми;
- test, що запускає усі тести за папки test;
- install — установка програмного забезпечення в локальний Maven-репозиторій, щоб зробити його доступним для інших проектів поточного користувача.
- deploy — стабільна версія програмного забезпечення поширюється на віддалений Maven-репозиторій, щоб зробити його доступним для інших користувачів.

- site — життєвий цикл генерації проектної документації [29].

3.7. Опис програмної реалізації

Проектування архітектури програмного забезпечення — це процес розроблення, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи [30]. Необхідно визначити цілі системи, вхідні та вихідні дані, розглянути усі компоненти, модулі та її структуру, а також зробити декомпозицію системи на підсистеми перед побудовою її ахітектури. Існує декілька методів проектування архітектури системи (стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з цих методів підходить по різному до побудови архітектури, а саме, до охарактеризуванню концептуальної, об’єктної й інших моделей за допомогою блок-схем, графів, структурних діаграм тощо. Проектування системи може здійснюватися на основі об'єкто-орієнтованого моделювання використовуючи UML, який використовує графічні позначення для створення абстрактної моделі системи [31-32].

Для розуміння поставленої задачі необхідно розробити діаграму послідовності, у якій показується взаємодія користувача з системою та внутрішня взаємодія в системі. На цій діаграму показан життєвий цикл сутності на часовій осі та її взаємодія з користувачем системи (актором).

На діаграмі послідовності об'єкти в основному представляють екземпляри класу або сутності, що володіють поведінкою, які об'єкти можуть виступати користувачами, а які ініціюють взаємодію, класи, що володіють поведінкою в системі або програмні компоненти, а іноді і системи в цілому.

Невід'ємною частиною об'єкта на діаграмі послідовності є лінія життя об'єкта. Лінія життя показує час, протягом якого об'єкт існує в Системі. Періоди активності об'єкта в момент взаємодії показуються за допомогою фокуса управління. Тимчасова шкала на діаграмі спрямована зверху вниз. На рисунку 3.8 зображена діаграма послідовності для роботи системи з мікросервісами та системою хмарних обчислень.

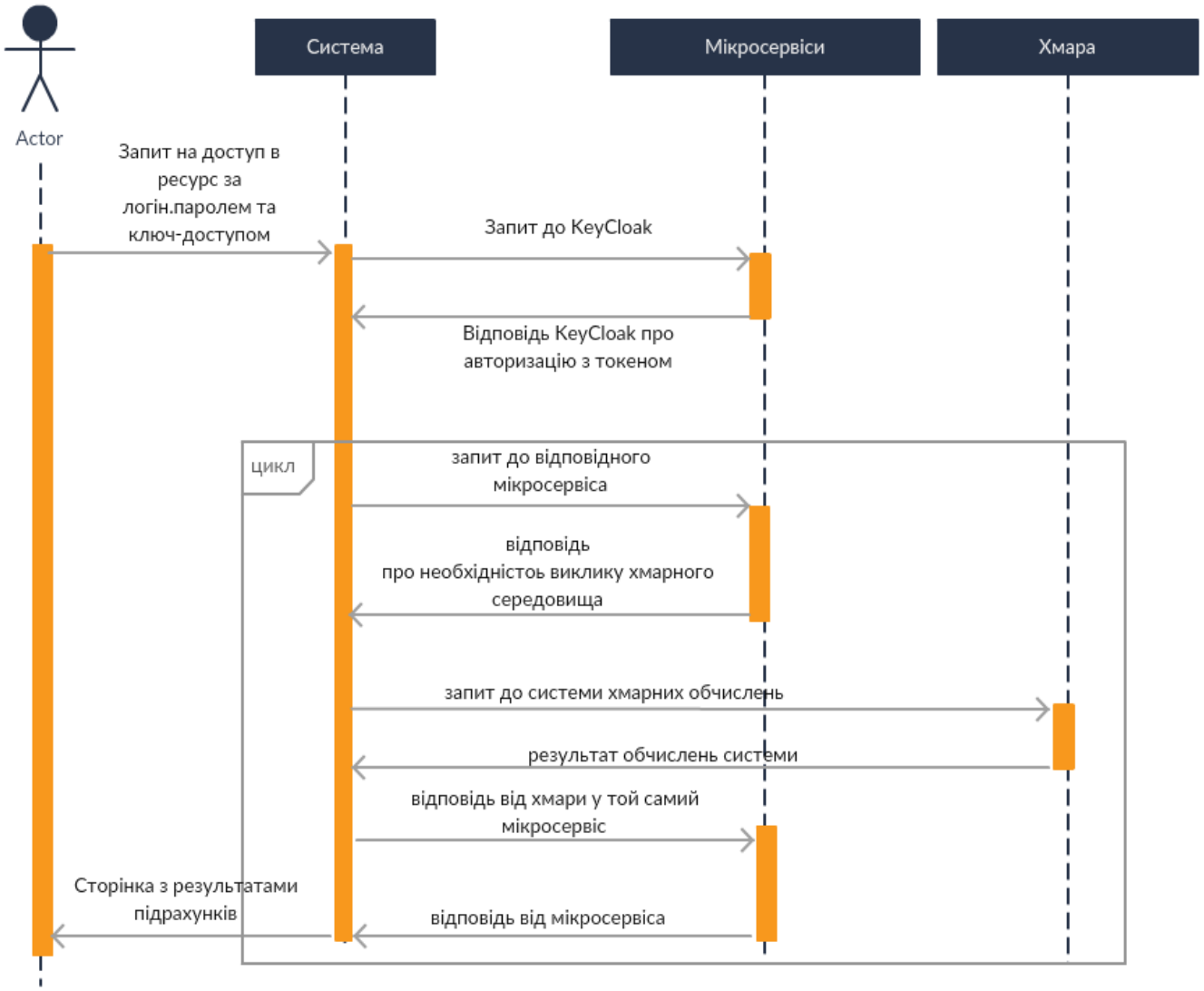

Рисунок 3.8 — Діаграма послідовності для системи з мікросервісною архітектурою в системах хмарних обчислень.

Висновки до розділу 3

У **третьому розділі** було:

1. Проаналізовано та обрано засоби реалізації для створенні програмного забезпечення;
2. Розроблено діаграму послідовності, яка представляє зв’язок користувача з функціоналом програмної системи;
3. Побудовано діаграму основних класів системи, яка дозволяє детальніше описати архітектуру програмного комплексу.

4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблений програмний продукт призначений для використання програмістами, адже він надає лише інструментарій для роботи мікросервісної архітектури в системах хмарних обчислень, тому необхідно в першу чергу розуміти як налаштувати даний застосунок в своїй системі. Розглянемо як сконфігурувати роботу даного застосунку з системою хмарних обчислень на прикладі Amazon Web Service. Для цього необхідно у файлі налаштувань template.yaml розробленого застосунку прописати дані роботи в AWS lambda (рисунок 4.1).

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  Offsite backup for DB, using pg_dump.

Globals:
  Function:
    Runtime: java8
    MemorySize: 512
    Timeout: 240
    Environment:
      Variables:
        IN_PRODUCTION: false
        TOKEN: goodToken


Resources:
  TermsAndConditions:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./target/lambda.jar
      Handler: com.insurancemenu.controller.AragController::paymentInfo
      Events:
        AragBackEndTestGET:
          Type: Api
          Properties:
            Path: /paymentInfo/{proxy+}
            Method: get
```

Рисунок 4.1. — Конфігурація налаштування роботи з Amazon Web Service

Після того як функція Lambda створена, вона перебуває в стані постійної готовності до запуску, подібно формулами електронних таблиць. Кожна функція містить користувальницький код і деякі дані конфігурації, включаючи ім'я функції і вимоги до ресурсів. Функції Lambda незберігають стан і ніяк не залежать від базової інфраструктури, тому Lambda може швидко завантажити стільки копій функції, скільки потрібно для масштабування відповідно до кількості вхідних подій.

Після завантаження коду в AWS Lambda можна зв'язати функцію з тими чи іншими ресурсами AWS, наприклад з кошиком Amazon S3, таблицею Amazon DynamoDB, потоком Amazon Kinesis або повідомленням Amazon SNS. При зміні стану ресурсу Lambda виконає функцію і налаштує обчислювальні ресурси для продовження обслуговування вхідних запитів [33].

Далі для роботи користувача з системою необхідно сконфігурувати KeyCloak, що відповідає за авторизацію користувачів, для цього необхідно встановити базу даних PostgreSQL. На рисунку 4.2 зображено конфігурацію для розгортання KeyCloak як мікросервіс.

```
@Bean
ServletRegistrationBean keycloakJaxRsApplication(KeycloakServerProperties keycloakServerProperties,
                                                 DataSource dataSource) throws Exception {

    mockJndiEnvironment(dataSource);

    EmbeddedKeycloakApplication.keycloakServerProperties = keycloakServerProperties;

    ServletRegistrationBean servlet =
            new ServletRegistrationBean(new HttpServlet30Dispatcher());
    servlet.addInitParameter( name: "javax.ws.rs.Application",
            EmbeddedKeycloakApplication.class.getName());
    servlet.addInitParameter(ResteasyContextParameters.RESTEASY_SERVLET_MAPPING_PREFIX,
            keycloakServerProperties.getContextPath());
    servlet.addInitParameter(ResteasyContextParameters.RESTEASY_USE_CONTAINER_FORM_PARAMS,
             value: "true");
    servlet.addUrlMappings(keycloakServerProperties.getContextPath() + "/*");
    servlet.setLoadOnStartup(1);
    servlet.setAsyncSupported(true);

    return servlet;
}
```

Рисунок 4.2. — Конфігурація налаштування KeyCloak як мікросервіса

Також для роботи KeyCloak необхідно підключити роботу за базою даних, для цього необхідно сворити сервер бази даних і для нього користувача і далі за логіном

та паролем при старті системи KeyCloak буде автоматично створювати підключення до вашої бази. На рисунку 4.3 зображено конфігурацію KeyCloak за підключеня до бази даних [34].

```
spring:
  datasource:
    username: ${IDMBROKER_DB_USER:postgres}
    password: ${IDMBROKER_DB_PASSWORD:root}
    driverClassName: org.postgresql.Driver
    driverDialect: org.hibernate.dialect.PostgreSQL82Dialect
    url: ${IDMBROKER_DB_URL:jdbc:postgresql://localhost/idmbroker}

server:
  port: 8088

resteasy:
  deployment:
    async_job_service_enabled: true


keycloak:
  server:
    contextPath: /auth
    adminUser:
       username: admin
       password: admin
```

Рисунок 4.3. — Конфігурація налаштування KeyCloak для роботи з базою даних.

Після даних конфігурацій та запуску сервіса KeyCloak, через його адмінпанель можна роботи багато налаштувань, а саме (рисунок 4.4):

- налаштувати основні дані для авторизації;
- додати та видалити користувачів;
- створити тарозподілити ролі користувачам;
- перевірити сесії користувачів.

Налаштувавши KeyCloak та Amazon Web Service необхідно перейти до налаштування власне розробленої системи. Для налаштування її вам не потрібно додаткого щось встановлювати, чи завантажувати, адже фреймворк побудований за

допомогою збірника пакетів Maven, що автоматично при запуску програми завантажує усі необхідні пакети.

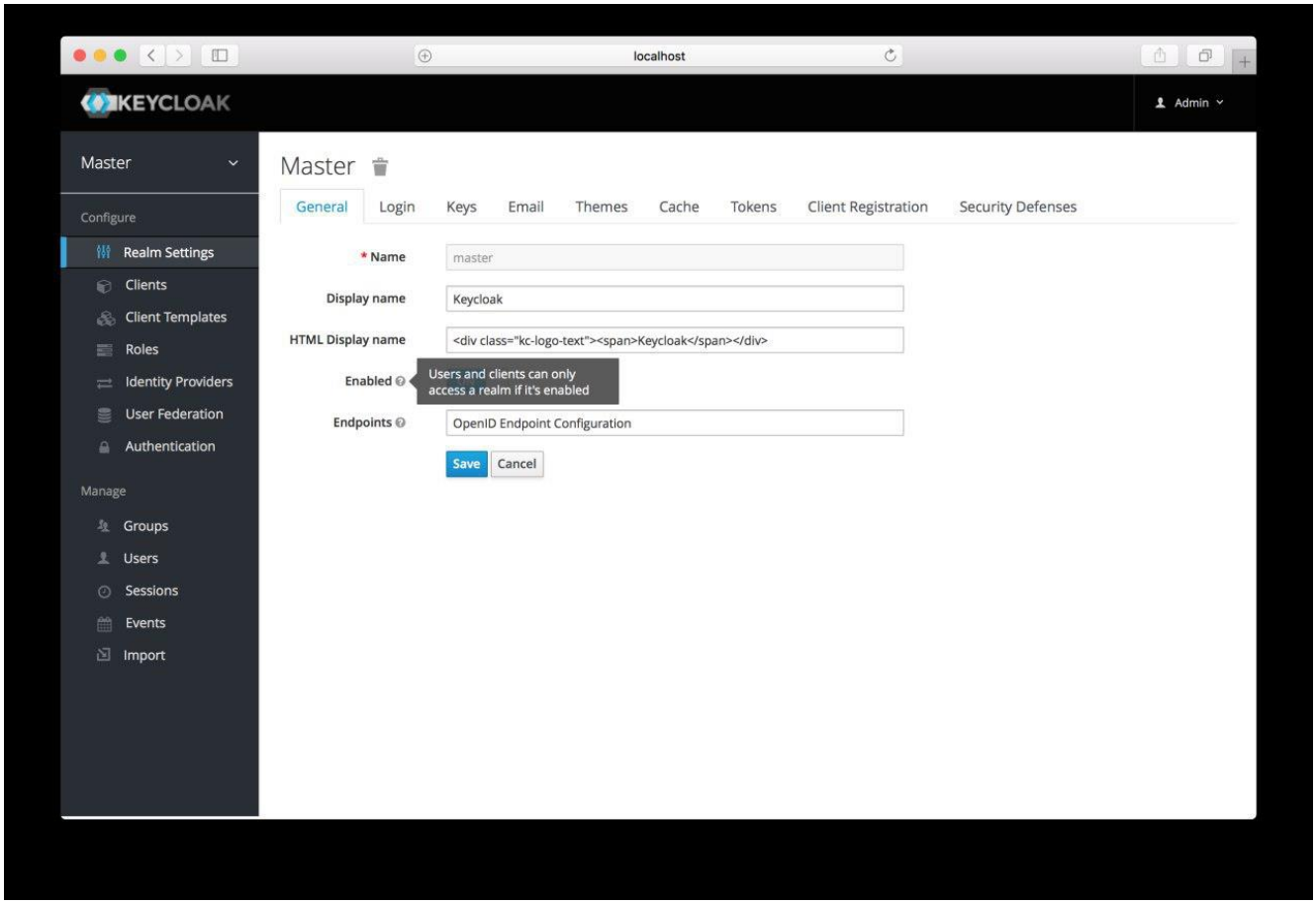

Рисунок 4.4. — Основне меню адмінпанелі KeyCloak.

Після всіх цих налаштувань залишається лише зареєструвати кожен мікросервіс в розробленому фреймворку, для цього в конфігураційний файл application.yaml у розділі “microservers” необіхдно додати посилання на ваші створені мікросервіси.

Висновки до розділу 4

1. Розроблено фреймворк для роботи систем з мікросервісною архітектурою в системах хмарних обчислень, спроектованого у попередньому розділі.

2. На прикладах розтлумачено, як конфігурувати даний фреймворк за іншими системами на конкретних прикладах.

5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

Ідея проекту полягає у створенні фреймворку для розроблення систем хмарних обчислень з мікросервісною архітектурою, що надає програмістам готовий набір функціоналу для організації роботи застосунку, що матиме систему авторизації користувачів по токенах, для взаємодії між мікросервісами та ключ-доступом для авторизації до хмарних обчислень. Такий фреймворк допоможе програмістам швидко побудувати архітектуру застосунка. Використання KeyCloak вирішує проблему аутентифікації користувачів у системах з мікросервісною архітектурою, забезпечуючи розшарування користувачів за рівнем доступу та забезпечує передачу токенів між мікросервісами. Дане програмне рішення буде корисним розробникам систем хмарних обчислень, що використовується для виконання високонавантежених задач.

5.1. Опис ідеї стартап-проекту

Проаналізуємо зміст ідеї, її можливі напрямки застосування, відмінність запропонована ідея від існуючих аналогів, а також основні переваги, які може отримати користувач системи. Результати аналізу представлені у таблиці 5.1.

Таблиця 5.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створення фреймворку, що вирішує проблему авторизації користувачів	Розроблення систем хмарних обчислень з мікросервісною архітектурою	Можливість швидко створити застосунок за рахунок, уже готового рішення проблеми авторизації

Наразі на ринку не існує стандартного рішення даної проблеми аутентифікації користувачів, через це, програмістам доводиться розробляти архітектуру авторизації користувачів у хмарних системах самостійно, а для цього потрібно мати глибоке розуміння їх роботи. Дане рішення буде корисним для тих програмістів, хто хоче розробити швидко програмне рішення не витрачаючи багато часу на інфраструктурні питання.

Доцільно провести аналіз потенційних техніко-економічних переваг ідеї. Результат аналізу у таблиці 5.2.

Таблиця 5.2. Визначення характеристик ідеї проекту

Техніко-економічні характеристики ідеї	Продукція конкурентів		Слабкі (W), нейтральні (N) та сильні (S) сторони		
Назва продукту	Розроблена система	Netflix OSS	W	N	S
Операційна система та версії	Кросс-платформенна	Кросс-платформенна		✓	
Мови програмування	Java	Java		✓	
Необхідність встановлення додаткового ПЗ	наявність Java SDK	наявність Java SDK		✓	
Відкритий вихідний код	Так	Ні			✓
Ціна	безкоштовний	$40 підписка			✓

Розроблений фреймворк являє собою кроссплатформенний застосунок, потребує мінімальну кількість додаткового програмного забезпечення. Розроблене програмне рішення має відкритий програмний код, що дозволяє гнучкіше налаштовувати систему.

5.2. Технологічний аудит ідеї проекту

Для проведення технічного аудиту ідеї проекту, необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту, визначити чи доступні ці технології, та чи потребують вони допрацювання. Результат представлений у таблиці 5.3.

Таблиця 5.3. Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Створення фреймворку, що вирішує проблему авторизації користувачів	Середовище розробки Intelij IDEA JetBrains	✓	Доступно (безкоштовна базова версія)
	Технологія KeyCloak	✓	Доступно (безкоштовне)
	Spring Framework	✓	Доступно (бескоштовно)
	Hibernate Framework	✓	Доступно (бескоштовно)

Таблиця 5.3. Технологічна здійсненність ідеї проекту (продовження)

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Створення фреймворку, що вирішує проблему авторизації користувачів	База даних PostgreSQL	✓	Доступно (бескоштовно)

Для реалізації ідеї проекту були обратні наступні технології:

- JetBrains 2018 — середовище розробки;
- Технологія KeyCloak — фреймворк автори зації користувачів;
- Spring Framework — фреймворк для розроблення веб додатків, що пропонує архітектуру застосунку та методи розробки, що роблять застосунок гнучким до змін;
- Hibernate Framework — фреймворк для роботи за базою даних;
- База даних PostgreSQL — база даних, що має досить гнучку структуру даних, а також являється однією з найшвидших;

Обрані технології є доступними, не потребують допрацювань, а також безкоштовні та надають усі необхідні можливості для реалізації поставленої задачі.

5.3. Аналіз ринкових можливостей запуску стартап-проекту

Аналіз ринкових можливостей, які можна використати для впровадження проекту на ринку, а також та аналіз ринкових загроз, через які реалізація проекту може ускладнитися, дозволяє передбачити можливі загрози та охарактеризувати найперспективніші напрямки розвитку проекту, спираючись на стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. Проведемо аналіз попиту (таблиця 5.4).

Таблиця 5.4. Попередня характеристика потенційного ринку стартап-проекту

Показники стану ринку	Характеристика
Загальна потреба в продукції	Необхідно, адже збільшується кількість систем, що використовують хмарні обчислення
Можливі річні обсяги випуску в натуральних показниках	До 50 копій
Річні обсяги випуску в вартісних показниках	500 – 1500$
Динаміка ринку (якісна оцінка)	Зростає
Наявність обмежень для входу	Специфіка ринку, адже фреймворк робиться для програмістів
Специфічні вимоги до стандартизації та сертифікації	Відсутні
Середня норма рентабельності в галузі (або по ринку)	79%

Ринок хмарних обчислень наразі показує дуже швидкі темпи зростання, адже дані системи дозволяють проводити складні та ресурсоємні обчислення, що, наприклад, необхідні для ще однієї сфери, що наразі набирає популярність, а саме — машине навчання. Архітектурний підхід з використанням мікросервісів, дозволяє розробити розподіленні системи, а також пришвидшити процес розробки програмного продукту та доставлення його до кінцевого користувача, через що, все більше нових систем розробляється за даною архітектурою.

Отже, проаналізувавши ринок, можна сказати, що ринок є досить привабливим для входження.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (таблиця 5.5).

Таблиця 5.5. Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія	Особливості поведінки споживачів	Вимоги споживачів до товару
Розробка системи хмарних обчислень з мікросервісною архітектурою	Програмісти	Програмісти не хочуть витрачати багато часу на налаштування інфраструктурних питань, адже вони зазвичай не приносять вигоди, порівняно з функціоналом, що може розробити за цей час програміст	• доступність; • відкритий вихідний код; • простота в налаштуванні.
	Викладачі університету	Дана категорія тісно взаємодіє зі студентами, де останні будуть приймати не опосередковану участь у розробці хмарної системи, а здебільшого у студентів, не на стільки глибокі знання у даній сфері, щоби самостійно розробити правильно	• доступність; • відкритий вихідний код; • простота в налаштуванні; • полегшення процесу розробки; • використання стандартної архітектури.

Визначення потенційної аудиторії сприяє дослідженню аналізу представленого ринкового середовища: аналіз факторів, сприятливих для успішного впровадження проекту, та аналіз перешкоджаючих факторів.

Результати представлені у таблицях 5.6 та 5.7 відповідно.

Таблиця 5.6. Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Поява конкурентів	Можлива поява конкурентів, які розроблять більш універсальну та гнучкішу систему	Розробка нового функціоналу покращення та оптимізація наявного
Зміни тенденцій ринку	Поява нових тенденцій до архітектури програмного забезпечення.	Практично неможлива ситуація, адже мікросервісна архітектура досить новий підхід, проте, вже зарекомендував себе. Адаптація наявного коду під нові стандарти, розширення функціоналу
Економічний спад	Зменшення попиту на створення систем хмарних обчислень, через зменшення купівельної спроможності	Зміна цільової аудиторії.
Зниження репутації компанії	Конкуренти розроблять систему, що буде більше підходити останнім тенденціям розвитку архітектури програмного забезпечення, або буде більш ефективно використовувати ресурси	Надання програмного засобу для того, щоб більше проектів розроблювалося за допомогою нього, а також вдосконалення вже існуючого

Таблиця 5.7. Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Відсутність прямих конкурентів	Даний застосунок являється першим фреймворком, що претендує на звання стандарту у підході до вирішення проблеми аутентифікації користувачів	Розповсюджувати створений продукт, проводити вдосконалення та розширення існуючого функціоналу
Відповідні тенденцій ринку	Застосунок розроблений за останніми тенденціями ринку IT сфери та допомагає в побудові додатків за мікросервісною архітектурою з використанням систем хмарних обчислень, що робить його привабливим для використовування	Розповсюджувати створений продукт, проводити вдосконалення та розширення існуючого функціоналу
Можливість побудови власної репутації	Новий «гравець» на ринку дає можливість зарекомендувати себе	Можливість зарекомендувати себе на ринку. Пошук клієнтів. Розповсюдження продукту поміж цільової аудиторії.

Надалі необхідно провести аналіз пропозиції – визначити загальні риси конкуренції на ринку, а саме визначити тип можливої майбутньої конкуренції та її інтенсивність, рівень конкурентоспроможності за рівнем конкурентної боротьби, видами товарів і галузевою ознакою (таблиця 5.8).

Таблиця 5.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії)
Тип конкуренції	Чиста Конкуренція напряму залежить від кількості конкурентів та якості продукції, що вони надають	Розповсюджувати створений продукт, проводити вдосконалення та розширення існуючого функціоналу
За рівнем конкурентної боротьби	Глобальна. Конкуренція не обмежується лише вітчизняним ринком	Прямою конкуренції не виявлено, а отже компанія, може виставляти ціну за власним бажанням та наробляти клієнтську базу.
За галузевою ознакою	Внутрішньогалузева Продукт є вузьконаправленним	Розроблений застосунок являє собою вузьконаправлений продукт, чим одразу займає нішу на ринку та має можливості для росту, шляхом збільшення та покращення функціоналу, а також створюватиме конкуренцію вже існуючим рішенням
Конкуренція за видами товарів	Марки-конкуренти Розроблений застосунок може мати конкурентів, які пропонують аналогічний функціонал	Розповсюджувати створений продукт, проводити вдосконалення та розширення існуючого функціоналу.

Таблиця 5.8. (продовження)

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії)
За характером конкурентних переваг	Цінова Важливо по якій ціні продається товар	Можливість збільшення вартості застосунку, при збільшенні функціоналу
За інтенсивністю	Марочна Можуть з’являтись конкуренти	Проводи рекламу та демонструвати розроблений застосунок на конференціях. Проводити вдосконалення та розширення існуючого функціоналу

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (таблиця 5.9) [35] - за моделлю п’яти сил М. Портера, який вирізняє п’ять основних факторів, що впливають на привабливість вибору ринку з огляду на характер конкуренції:

- конкурент, що вже є у галузі;
- потенційні конкуренти;
- наявність товарів-замінників;
- постачальники, що конкурують за ринкову владу;
- споживачі, які конкурують за ринкову владу.

За результатами аналізу робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспроможності.

Таблиця 5.9. Аналіз конкуренції в галузі за М.Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Клієнти	Товари-замінники
Складові галузі	Розробники аналогічних застосунків	Продукти з кращою оптимізацією, ширшим функціоналом	Основний фактор успішності в галузі, їх кількість важливіша за термін використання	Відсутні. Немає прямих конкурентів
Висновки	Інтенсивність конкурентної боротьби з боку прямих конкурентів відсутня	Перешкод для виходу на ринок немає. Прямих конкурентів немає. Строки виходу на ринок – один день	Необхідно розповсюдження в колах програмістів, тим самим здобуває нових клієнтів	Немає обмежень

За проведеним аналізом конкуренції за М. Портером, зазначеного у таблиці 5.9, а також беручи до уваги характеристики ідеї проекту (таблиця 5.2), вимог споживачів товару, представленого у таблиці 5.5 та факторів маркетингового середовища (таблиці 5.6 і 5.7) можна зазначити перелік факторів конкурентоспроможності (таблиця 5.10) [35].

Таблиця 5.10. Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	Обґрунтування
Відсутність прямих конкурентів на ринку	Початкова націленість на вітчизняний ринок для розповсюдження поміж програмістів, які потім можуть запроваджувати даний застосунок уже в іноземних компаніях, адже працюють на експорт

Таблиця 5.10. Обґрунтування факторів конкурентоспроможності (продовження)

Фактор конкурентоспроможності	Обґрунтування
Доступність створеного продукту (програмно)	Розроблений застосунок має мінімальні системі потреби, адже націлена на використання потужностей хмарний систем
Легкість і простота конфігурації	Має відкритий програмний код, що робить систему гнучкою для зміни
Використання технологій, що не залежить від середи запуску	При розробці застосунку були використані технології, що являються кроссплатформеними
Необхідність у підтримці	Застосунок не потребує підтримки, адже являється інструментом розробки
Додаткові компоненти	Немає необхідності встановлювати будь яке додаткове програмне забезпечення

Після визначення факторів конкурентоспроможності проводиться аналіз сильних та слабких сторін стартап-проекту, проведений у таблиці 5.11.

Таблиця 5.11. Порівняльний аналіз сильних та слабких сторін проекту

Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів						
		–3	–2	–1	0	+1	+2	+3
Мала кількість / відсутність конкурентів	10				✓			
Системні вимоги	20			✓				
Простота використання	19	✓						
Не потрібен супровід	11					✓		

Для отримання повної картини ринкового аналізу можливостей впровадження проекту необхідно є провести SWOT-аналізу спираючись на охарактеризовані вище ринкомі можливості та загрози, а також сильні та слабкі сторони, тобто розписати матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 5.12).

Таблиця 5.12. SWOT-аналіз проекту

• Сильні сторони (S): • доступна ціна; • новітні технології; • свіжий погляд на проблему; • гнучкість до змін;	• Слабкі сторони (W): • брак матеріальної бази; • недостатність первинного фінансування; • високий поріг входження до проекту; • відсутність репутації компанії;
• Можливості (O): • специфічний функціонал за додаткову плату; • освоєння нових ринків збуту; • масовість продукту; • тісна співпраця з адміністраторами систем хмарних обчислень.	• Загрози (T): • поява конкурентів; • зниження репутації компанії; • економічний спад; • зміни тенденцій попиту.

На основі SWOT-аналізу розробимо альтернативу ринкової поведінки для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок.

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (таблиця 5.13).

Таблиця 5.13. Альтернативи ринкового впровадження стартап-проекту

Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
Вихід на нові ринки	Пошук інвесторів	5-6 місяців
Тісна співпраця з конкретною вітчизняною компанією	Висока	Після заключення договору від 3 до 6 місяців

Отже, спираючись на дані проведеного аналізу, можна зробити висновок, що для початку необхідно вивести компанію на вітчизняний ринок, а далі шукати інвесторів та розвивати систему під конкретних користувачів.

5.4. Розроблення ринкової стратегії проекту

Розробка ринкової стратегії перш за все передбачає визначення стратегії охоплення ринку [35], включаючи опис цільових груп потенційних споживачів, які визначені у таблиці 5.14.

Таблиця 5.14. Вибір цільових груп потенційних споживачів

Опис цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в сегменті	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Програмісти	Потребують	Попит є	Незначна	Просто
Викладачі університетів	Потребують	Попит є	Незначна	Помірно
Адміністратори систем хмарних обчислень	Потребують	Попит є, проте менший ніж у попередніх	Незначна	Складно

Зважаючи на те, що розроблений застосунок розроблений для досить вузької аудиторії, а отже цільові групи мають невеликі відмінності. Оскільки компанія хоче отримати перші прибутки як найшвидше, то можна зробити висновок, що провести масовий маркетинг доцільно враховувати таку особливість цільової групи, тобто мати одну єдину спрямованість розвитку, а вже потім виходити на більш широкі ринки.

Після проведення аналізу потенційних груп споживачів (сегментів), необхідно вибрати цільові групи для яких уже визначається маркетингова стратегія, проводиться реклама спрямована саме на нею, враховуючи специфіку її та визначається стратегію охоплення ринку.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яка визначається у таблиці 5.15.

Таблиця 5.15. Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Вихід на нові ринки	Стратегія диференціації	Надання програмного застосунку, який містить весь необхідний функціонал та немає помилок	Стратегія диференціації
Розширення функціоналу	Стратегія диференціації (допускається стратегія спеціалізації)	Надання кращої реалізації певних аспектів, порівняно з конкурентами	Стратегія диференціації (допускається стратегія спеціалізації)

Вибір стратегії конкурентної поведінки визначається у таблиці 5.16.

Таблиця 5.16. Визначення базової конкурентної поведінки

Чи є проект «першопроходцем» на ринку	Так
Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Так
Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Ні
Стратегія конкурентної поведінки	Стратегія виклику лідера

Визначивши базову стратегію розвитку та стратегію базової конкурентної поведінки можна сформувати основні вимоги до цільової групи споживачів (сегментів), та постачальника (стартап-компанії) можна розробити розробити стратегію позиціонування (таблиця 5.17) програмного застосунку. Ідея даної стратегії полягає у формуванні ринкової позиції, за яким споживачі мають ідентифікувати торгівельну марку або проект .

Таблиця 5.17. Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
Доступна ціна, простота і зручність використання	Стратегія диференціації	відкритий вихідний код; простота в налаштуванні; полегшення процесу розробки;	Потужність розробленого застосунку; відмовостійкість

Отже, за результати проведених аналізів, можна сформувати роботу стартап-компанії на ринку, а саме — за стратегією диференціації, що передбачає необхідність підтримувати якість програмного застосунку, дотримуючись у конкурентній поведінці стратегії «виклику лідера», тобто випускається один товар для усіх можливих споживачів.

5.5. Розроблення маркетингової програми стартап-проекту

Першим кроком під час розробки маркетингової програми стартап-проекту є формування маркетингової концепції товару [35]. Для цього у таблиці 5.18 підсумовані результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18. Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
Оцінка якості ПП	Пришвидшення розробки програмного забезпечення. Стандартизація архітектури застосунку	Розроблений програмний продукт, вирішує проблеми аутентифікації користувачів у системах з мікросервісною архітектурою. Функціонал для взаємодії з системами хмарних обчислень

Необхідно розробити трирівневу маркетингову модель товару, що передбачає уточняються ідея продукту, його фізичні складові та особливості процесу його надання (таблиця 5.19) [35].

Таблиця 5.19. Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
Товар за задумом	Розробити застосунок, що дозволить використовувати архітектуру мікросервісів у системах фмарних обчислень, тим самим вирішуючи проблему аутентифікації та запроваджуючи рішення, що буде корисним програмістам у розробці їх власного програмного забезпечення.
Товар у реальному виконанні	Властивості/характеристики
	Реалізовано фреймворк, що надає функціонал взаємодії застосунка з мікросервісною архітектурою з системою хмарних обчислень
Товар із підкріпленням	До продажу: застосунок, що має набір функціоналу для взаємодії з системами хмарних обчислень. Після продажу: Фреймворк адміністрації систем високонавантаженних та високопродуктивних систем хмарних обчислень.

Розроблений програмний застосунок без первинних налаштувань не може працювати, а отже спочатку існує необхідність у тому щоб розібратися у розробленому фремворку, а далі вже застосовувати при розробці власної системи. Можна розробити і власноруч, проте на це необхідно буде затратити більше часу, ніж розібратися, як сконфігурувати і використати запропоноване рішення. Розробка фреймворку, що вирішує проблему аутентифікації користувачів у системах хмарних обчислень з використання мікросервісної архітектури являє собою наукову новизну, тобто те, що не має аналогів, а тому є необхідність у фіксуванні авторських прав або отриманні патенту.

Для того, щоб визначити цінові рамки розробленого застосунку, спрогнозувати купівельну спроможність потенційних покупців необхідно провести аналіз ціни на

товари-аналоги, а також аналіз рівня доходів цільової групи споживачів описано в таблиці 5.20.

Таблиця 5.20. Визначення меж встановлення ціни

Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни
500 – 700 $	300 – 3000 $	10 – 15 $

Важливим пунктом у реалізації стартап-проектів є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 5.21):

- Проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);
- Вибір та обґрунтування оптимальної глибини каналу збуту;
- Вибір та обґрунтування виду посередників [35].

Таблиця 5.21. Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Бажання отримати більше за менші гроші та швидше готовий функціонал, що веде до швидшого повернення інвестицій	Пошук клієнтської бази та продаж,	Нульовий рівень: тільки виробник	Вертикальна маркетингова система

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 5.22).

Таблиця 5.22. Концепція маркетингових комунікацій

Поведінка цільових клієнтів	Канали комунікацій цільових клієнтів	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення
Бажання отримати більше за менші гроші	Будь-які	✓ низька ціна; ✓ легкий і простий у конфігурації; ✓ гнучкість до змін; ✓ відкритий програмний код	Донести до користувача суть продукту, його якість, та залучити якомога більше зацікавлених клієнтів

Висновки до розділу 5

Спираючись на проведений аналіз ринку та конкурентноспроможності розробленого стартапу можна розробити маркетингову програму, виокремити конкуренті переваги та спрогнозувати перспективи проекту.

Отже, маркетингова програма має бути побудована наступним чином:

1. Розробка функціоналу програмного застосунку;
2. Аналіз затребуваності товару на ринку та визначення цільової аудиторії;
3. За базову стратегію розвитку необхідно використати стратегію диференціації, яка передбачає, що конкурентноспроможність продукту досягається шляхом, що продукт має якнайкраще відповідати потребам споживачів, що в свою чергу досягається ретельним вивченням ринку. Споживацька цікавість має досягатися завдяки одній або декільком відмінних та інноваційних характеристик;
4. Стратегія конкурентної поведінки передбачає використання стратегії лідера, тобто розроблений застосунок орієнтується на всіх ймовірних споживачів на споживчому ринку, у тому числі і клієнтів конкуруючих фірм. Основною цілю є закріплення на ринку та згодом випередження лідерів обраного сегменту.

Конкурентні переваги розробленого застосунку очевидні, адже прямих аналогів не виявлено, а отже і конкурентів у продукту немає. Саме тому, перспектива

впровадження даної розробки на ринку оцінюються як дуже високий. Системи хмарних обчислень набирають все більшу популярність з кожним днем, тому можлива поява прямих конкурентів, тому розроблений застосунок необхідно постійно вдосконалювати, додавати новий функціонал та оптимізовувати старий, тим самим збільшуючи цільову аудиторію користувачів.

ВИСНОВКИ

1. На основі аналізу сучасних методів авторизації було виявлено, що безпосередня авторизація у програмах з мікросервісною архітектурою у системах хмарних обчислень.

2. Аналіз програмних комплексів для авторизації показав, що не існує програмних засобів, які дозволяють використовувати мікросервісну архітектуру для використання в системах хмарних обчислень.

3. Було проаналізовано архітектуру сучасних систем хмарних обчислень і на основі цих даний розроблено концепт реалізації ідеї взаємодії із мікросервісами

4. Розроблено архітектуру підходу для подолання проблеми авторизації.

5. Проаналізовано та обрано засоби реалізації для створенні програмного забезпечення.

6. Розроблено систему авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою

7. На основі поданої ідеї було розроблено бізнес-стартап проекту. Був проведений технологічний аудит, проведено аналіз ринкових можливостей, розроблена базова стратегія розвитку програмного продукту, маркетингова програма, стратегія конкурентної поведінки на ринку, розглянуто перспективи впровадження з огляду на потенційні групи користувачів програмного продукту.

8. Подальші дослідження можуть бути спрямовані на розширення функціоналу запропонованого рішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler M. Production Ready Microservices / Martin Fowler. – 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2016. – 172 с.
2. Newman S. Building Microservices / Sam Newman. – 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2015. – 473 с.
3. Escoffier C. Building Reactive Microservices in Java / Clement Escoffier. – 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2017. – 83 с.
4. Thomas D. The Current State Of Authentication: We Have A Password Problem. Smashing Magazine 2016.
5. Сміт. Р. Аутентифікація: від паролей до відкритих ключів (Authentication: From Passwords to Public Keys First Edition.) / Річард Э. Сміт — М.: Вильямс, 2002. — 432 с.
6. Microsoft. Basic Authentication [Електронний ресурс] / Microsoft. – 2005. – Режим доступу: https://docs.microsoft.com/en-us/windows/desktop/secauthn/basic-authentication-concepts Дата доступу: 27.10.2018
7. Офіційна документація Amazon Web Service [Електронний ресурс]/ Amazon — Режим доступу: https://docs.aws.amazon.com/en_us/cli/latest/userguide/cli-chap-getting-started.html Дата доступу: 27.10.2018
8. Duncan D. "Two-factor authentication" / de Borde Duncan., 2009.
9. Boyd R. Getting Started with OAuth 2.0. / Ryan Boyd — 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2012. — 56 с.
10. Cantor S. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 / Scott Cantor — Errata Composite: 2015. - 93
11. Horrow S., Gupta S., Sardana A., Abraham A.: Secure Private Cloud Architecture for Mobile Infrastructure as a Service / Susmita Horrow, Sanchika

Gupta, Anjali Sardana, Ajith Abraham — Department of mathematics IIT Roorkee, 2012.

12. Atkisson B. How Red Hat re-designed its Single Sign On (SSO) architecture, and why.. / Brian Atkisson, Red Hat. 2018. [Електронний ресурс] — Режим доступу https://developers.redhat.com/blog/2016/10/04/how-red-hat-re-designed-its-single-sign-on-sso-architecture-and-why/ Дата доступу: 09.12.2018
13. Youseff L. Toward a Unified Ontology of Cloud Computing / L. Youseff, M. Butrico, D. Da Silva. — Santa Barbara, CA 93106,: University of California, 2008.
14. Buyya R, Ranjan R and Calheiros R. Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities/ Rajkumar Buyya, Rajiv Ranjan and Rodrigo N. Calheiros — Grid Computing and Distributed Systems(GRIDS) Laboratory Department of Computer Science and Software Engineering The University of Melbourne, Australia, 2009.
15. Офіційна документація KeyCloak. [Електронний ресурс]. – Режим доступу : https://www.keycloak.org/documentation.html Дата доступу: 09.12.2018
16. Фаулер М. Рефакторинг. Улучшение существующего кода [Електронний ресурс] / М. Фаулер. — Пер. с англ. — СПб: Символ-Плюс, 2003. — 432 с.
17. И.Н. Блинов, В.С. Романчик Минск: издательство «Четыре четверти», 2013. — 896 с.
18. Шеффер К., Хо К., Харроп Р. Spring 4 для профессионалов (Pro Spring 4) / Кріс Шеффер, Кларенс Хо, Роб Харроп. — М.: «Вильямс», 2017. — 752 с.
19. Офіційний сайт Spring Framework [Електронний ресурс] — Режим доступу: https://spring.io/ - Дата доступу: 27.10.2018
20. Walls C. Spring in Action / Craig Walls., – 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2018. — (5) – 520 с.
21. Документація Spring Framework. 1. Introduction to Spring Framework [Електронний ресурс] — Режим доступу:

https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch01s02.html Дата доступу: 27.10.2018

22. Документація Spring Framework. 6. The Web [Електронний ресурс] — Режим доступу: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html Дата доступу: 27.10.2018
23. Документація Spring Data Framework [Електронний ресурс] — Режим доступу:https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ Дата доступу: 27.10.2018
24. Офіційний сайт Spring Data Framework [Електронний ресурс] — Режим доступу: http://spring-projects.ru/projects/spring-data/jpa/ Дата доступу: 27.10.2018
25. Документація Spring Data Framework [Електронний ресурс] — Режим доступу:https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ Дата доступу: 27.10.2018
26. James E. Hibernate: A Developer's Notebook / Elliott James. – 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2004. – 190 с.
27. Офіційний сайт Hibernate Framework [Електронний ресурс] — Режим доступу: http://hibernate.org/orm/ Дата доступу: 27.10.2018
28. Van Zyl J. Maven: Definitive Guide/ Jason Van Zyl. – 1005 Gravenstein Highway North, Sebastopol, CA 954: O'Reilly Media, Inc, 2008. – 469 с.
29. Документація Maven Framework [Електронний ресурс] — Режим доступу: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference Дата доступу: 27.10.2018
30. Чернецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. — Издательский дом «Питер», 2005. — 730 с.
31. Кознов Д.В. Языки визуального моделирования: проектирование и визуализация программного обеспечения. Учебное пособие СПб.: Изд-во СПбГУ, 2004, 143 с.

32. D. Gracanin. Software Visualization. Innovation in Systems and Software Engineering / Gracanin D., Matkovic K., Eltoweissy M. - A NASA Journal. V. 1, № 2, September 2005, Springer, p. 221-230.
33. Amazon introduces Lambda, Containers at AWS. // SD Times. – 2014.
34. К. Дж. Дейт. Введение в системы баз данных / An Introduction to Database Systems. — 7-е изд. — «Вильямс», 2001. — 1092 с.
35. Розроблення стартап-проекту [Електронний ресурс] : Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей / За заг. ред. О.А. Гавриша. – Київ : НТУУ «КПІ», 2016. – 28 с.

ДОДАТОК А

Апробації

УКР.НТУУ “КПІ”.ТР32269_19М

Аркушів 11

2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVI Міжнародної науково-практичної конференції аспірантів, магістрантів і студентів м. Київ, 24-27 квітня 2018 року,

ТОМ 2

Київ- 2018

УДК 004.4

Магістрант 5 курсу, гр. ТР-71мп Прижков А.О.
Доц., к.т.н. Смаковський Д.С.

ЗАХИСТ СЕРЕДИ ХМАРНИХ ОБЧИСЛЕНЬ ШЛЯХОМ ВЕРИФІКАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА НАЯВНІСТЬ ДЕСТРУКТИВНИХ ВЛАСТИВОСТЕЙ

З початком інформаційної епохи стала зростати потреба суспільства обробляти все більші обсяги інформації, а також надавати доступ до даних в довільний момент часу. Для вирішення цього завдання використовується підхід, що забезпечує збереження інформації з використанням середовищ хмарних обчислень. Середа хмарних обчислень - це програмно-апаратна модель засобів обчислювальної техніки, що дозволяє отримувати віддалений доступ до обчислювальних ресурсів в будь-який момент часу [1]. Проте, при використанні середовищ хмарних обчислень зустрічаються з проблемою виявлення програмного забезпечення, яка не є шкідливим, але містить в собі помилки розробника, які можуть привести до деструктивному впливу на середовища хмарних обчислень [2].

Для вирішення цієї проблеми було вирішено створити застосунок, що реалізовує методику, яка дозволяє виділяти набір деструктивних властивостей програмного забезпечення, верифікацію на відсутність яких потрібно проводити [3]. За допомогою цієї методики можливо виділити набори програмних інструкцій, виявлення яких дозволить назвати програмне забезпечення некоректним для конкретного середовища хмарних обчислень. Наприклад, не проводячи перевірки програмного забезпечення на помилки втрати пам'яті, некоректну роботу з обчислювальними або ємнісними ресурсами, можливо назвати програмне забезпечення некоректним, якщо в уже згадуваному програмному забезпеченні присутні виклики певних програмних інструкцій.

Подібний підхід дозволить постачальнику послуг середовища хмарних обчислень самостійно обирати набір деструктивних властивостей програмного забезпечення, аналіз на наявність яких потрібно зробити перед його виконанням в середовищі хмарних обчислень. Також запропонований підхід дозволить не проводити повну формальну верифікацію коректності роботи програмного забезпечення, що займає тривалий час, вимагає залучення окремих фахівців і неможливо в автоматизованому режимі.

Для реалізації даної задачі було запропоновано використати мову програмування Java, що дозволяє працювати із застосуванням різних платформ. Для розробки інтерфейсу користувача було запропоновано використати платформу JavaFX. Застосунок дає можливість виконувати наступні функції: завантаження програмного забезпечення для аналізу, проведення аналізу на наявність деструктивних властивостей, а також робити звіт проведеного аналізу.

Перевагою даної системи є те, що немає необхідності виконувати програмне забезпечення для виявлення деструктивних властивостей, а аналізується лише код програмного засобу.

Перелік посилань:

1. The NIST Definition of Cloud Computing – Режим доступу: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145 .pdf Дата доступу: 19.03.2018
2. Threat Classification – Режим доступу: http://projects.webappsec.Org/w/page/13246978/Threat Classification Дата доступу: 19.03.2018
3. Филоненко А. В., Исаев И. К., Сидоров Д. В., Туманов Ю. М.. Поиск уязвимостей по бинарному коду с помощью проверки выполнимости ограничений. //Безопасность информационных технологий 2010. №2. С.83-86

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»
Теплоенергетичний факультет

Київська державна академія водного транспорту
імені П.Конашевича-Сагайдачного

Інститут кібернетики ім.В.М.Глушкова НАН

V науково-практична дистанційна конференція
молодих вчених і фахівців
з розробки програмного забезпечення

«СУЧАСНІ АСПЕКТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

15 травня 2018

м. Київ

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Національний університет “Києво-Могилянська академія”
Вища економіко-гуманітарна школа (Польща)

СТАЛИЙ РОЗВИТОК — XXI СТОЛІТТЯ: УПРАВЛІННЯ, ТЕХНОЛОГІЇ, МОДЕЛІ

Дискусії 2018

Колективна монографія

Київ, Україна
2018

Система авторизації мікросервісів на основі KeyCloak для захисту середовища хмарних обчислень

к.т.н. Смаковський Д.С., Прижков А.О.

Розглянуто спосіб побудови системи авторизації користувачів у системі хмарних обчислень з мікросервісною архітектурою. Запропоновано рішення побудови фреймворку, для аутентифікації користувачів на основі KeyCloak.

Рассмотрен способ построения системы авторизации пользователей в системе облачных вычислений с микросервисною архитектурой. Предложено решение построения фреймворка для аутентификации пользователей на основе KeyCloak.

Ключові слова: мікросервіси, система авторизації, хмарні обчислення, захист.

Актуальність проблеми.

Термін “мікросервісна архітектура” з'явився протягом останніх кількох років, щоб описати спосіб проектування програмних застосувань як наборів самостійно розгорнутих сервісів. Хоча точного визначення цього архітектурного стилю не існує, існують певні загальні характеристики: організації сервісів навколо бізнес потреб, автоматичне розгортання, перенесення логіки від шини повідомлень до приймачів, децентралізований контроль над мовами програмування та даними». (М. Фаулер) [1].

Даний архітектурний підхід здобуває все більшу популярність, адже він надає ряд переваг порівняно з монолітною архітектурою, а саме: підтримувати невеликі сервіси легше; можливість горизонтального масштабування; відмовостійкість коду; незалежність вибору технологій для кожного з сервісів.

Проте, цей підхід має один суттєвий недолік, для середовищ хмарних обчислень, адже для них зазвичай використовують аутентифікацію по ключам-доступам (наприклад, найпопулярніше хмарне середовище Amason Web Service використовує саме цей тип), а для мікросервісів необхідно використовувати авторизацію користувача лише по токенах, адже потрібно передавати цей токен між усіма мікросервісами. Отже, постає проблема, що нам потрібно передавати два типи унікальних даних, що збільшує розмір запиту і при великій кількості запитів, це може спричинити відмову системи, а також, в обох цих засобах аутентифікації користувачів використовується їх логін та пароль, що є дубліюванням даних.

Новизна.

На сьогоднішній день вже розроблені фреймворки для побудови функціоналу авторизації користувачів у системах з мікросервісною архітектурою, проте немає фреймворку, що дає змогу розробити процес аутентифікації для хмарних систем обчислень і при цьому використовувати мікросервісну архітектуру. Розроблений фреймворк буде розробленим на основі фреймворку для авторизації KeyCloak [2]. Розроблений Фреймворк розширює токен для того, щоб об’єднати інформацію про необхідну для авторизації користувача у мікросервісах, а також для авторизації у середовищах хмарних обчислень.

Основна частина

Стиль мікросервісов - це підхід, при якому єдине додаток будується як набір невеликих сервісів, кожен з яких працює у власному процесі і взаємодіє з іншими використовуючи легковажні механізми, Як правило, взаємодія організована на основі протоколу передачі даних HTTP. Ці сервіси побудовані навколо бізнес-потреб і

розгортаються незалежно з використанням повністю автоматизованої середовища. Існує абсолютний мінімум централізованого управління цими сервісами. Самі по собі ці сервіси можуть бути написані на різних мовах і використовувати різні технології зберігання даних. Зазвичай, взаємодія між мікросервісами відбувається у вигляді REST запитів, що і робить кожен сервіс незалежним один від одного у виборі технологій.

Розглянемо переваги цього підходу порівняно з монолітним застосунком:

- підтримувати невеликі сервіси легше, а також у разі необхідності внесення змін, не потрібно перезбирати увесь застосунок, а лише даних сервіс, що пришвидшує оперативність введення змін у роботу в рази;
- горизонтальне масштабування не викликає труднощів, тоді як у монолітному підході, воно майже неможливе, а в теперішній час, коли кількість користувачів інтернету на різними сервісами збільшується постійно, неможливість розподілення навантаження на сервіси призведе до відмови сервісів, тому зі збільшенням користувачів системи, для адміністрування монолітного проекту, потрібно набагато більше коштів;
- відмовостійкість мікросервісного застосунку набагато вища, адже при відмові одного з екземплярів сервісів, його образу може замінити інший, тоді як у монолітному, необхідно змінити весь екземпляр застосунку.

Чим відрізняється аутентифікації і авторизації. Коли ми говоримо, що додаток захищений — це означає, що у нього є ресурси, доступ до яких обмежений наявністю певних прав доступу. Щоб ці права отримати ми повинні пройти процес аутентифікації(підтвердження, що користувач є тим, за кого себе видає). Простим прикладом аутентифікації є форма введення логіна і пароля. Тобто користувачів вводити свій ідентифікатор (або логін) і підкріплює його паролем, тим самим доводячи, що цей ідентифікатор дійсно належить йому. Далі система на основі ідентифікатора користувача знаходить ті права, які цьому користувачеві призначені. Як ці правила задаються і звідки система їх отримує не так важливо. Варіантів тут може бути безліч, серед яких простий текстовий файл, реляційна БД, LDAP або окремий сервер авторизації. Останній крок — це порівняння необхідних прав доступу ресурсу, з правами конкретного користувача. Він називається процесом авторизації.

KeyCloak надає простий адміністративний інтерфейс для налаштування рівнів безпеки в веб-додатках. Він дозволяє швидко захистити додаток через форму логіна-пароля, відокремити управління користувачами і правами від логіки додатки, організувати SSO. Роздивимося процес аутентифікації в KeyCloak (рисунок 1):

- Крок 1: Запит захищеного ресурсу. Користувач в браузері звертається по URL до закритого ресурсу;
- Крок 2: Закрите додаток перенаправляє неавторизованого користувача на сервер аутентифікації KeyCloak;
- Крок 3: KeyCloak відображає сторінку аутентифікації (логін / пароль, соціальний логін, і т.д.);

- Крок 4: Користувач проходить етап аутентифікації. Для простоти будемо вважати, що вводить логін і пароль;
- Крок 5: KeyCloak видає тимчасовий токен (секрет) і робить перенаправлення на сторінку захищеного ресурсу;
- Крок 6 і крок 7: Додаток перевіряє валідність тимчасового токена і змінює тимчасовий на постійний JWT токен;
- Крок 8: На захищеному додатку проходить етап формування контексту безпеки. Користувачеві відображається захищений ресурс;

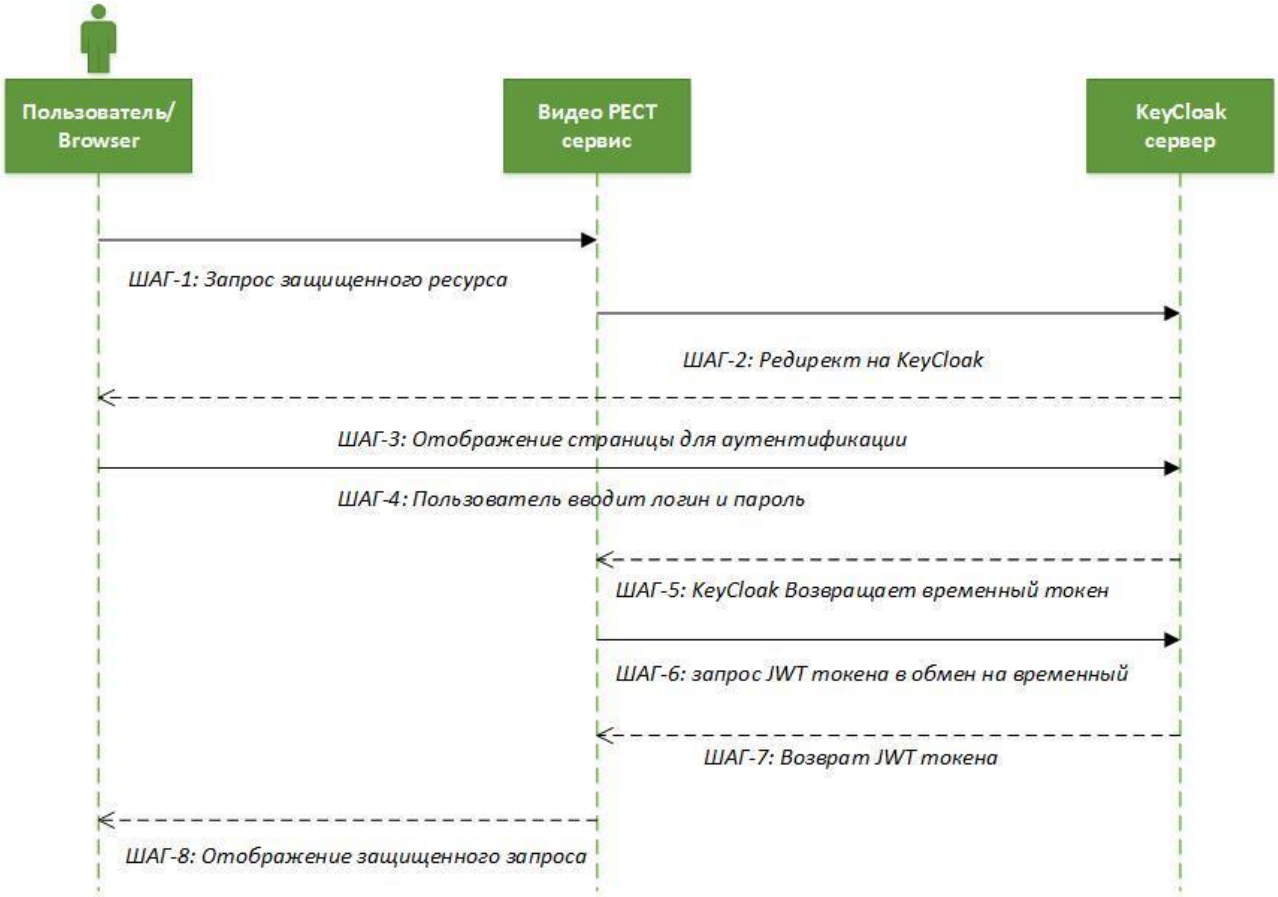

Рисунок 1. процес аутентифікації в KeyCloak

JWT (JSON Web Token) — це стандарт, який визначає компактний і автономний спосіб для захищеної передачі інформації між сторонами у вигляді JSON-об'єкта.

Основні властивості:

- Компактний. На відміну від SAML повідомлень (на основі XML), формат JWT виглядає набагато простіше. Складається з трьох частин: заголовок, основна інформація і цифровий підпис;
- Ємний. Містить інформацію по аутентифікованим користувачеві, включаючи ролі;
- Самодостатній. Для перевірки токена не потрібно звертатися до єдиного сервера (сервера idP, сервісу sts). Цю перевірку додаток може проводити самостійно, маючи в наявності відкритий ключ [3].

Відповідно до стандарту токен складається з трьох частин в base-64 форматі, між якими ставлять крапку. Перша частина називається заголовком (header), в якій міститься тип токена і назва хеш-алгоритму для отримання цифрового підпису. Друга частина зберігає основну інформацію (користувач, атрибути, ролі і т.д.). Третя частина - цифровий підпис.

Підсумовуючи, KeyCloak видає токен, що дозволяє користувачеві робити запити на мікросервіси та відповідно їх рівні доступу отримувати первую інформацію, проте, даний токен не дозволяє пройти авторизацію у системі хмарних обчислень. Саме цю проблему і вирішує розроблений фреймворк — створює обгортку для роботи з сервісом аутентифікації користувачів. Процес взаємодії користувача з нашим фреймворком. Користувач на сторінці авторизації вводить логін, пароль та додає свій ключ доступу. Уся форма потрапляє до нашого фреймворку, де далі відбувається запит до KeyCloak, а потім до середовища хмарних обчислень, щоб перевірити доступ. Даний фреймворк зберігає усю інформацію про користувача в кеші, тим самим не потребує серверних потужностей. Оскільки всі запити проходять через даний фреймворк, то при потребі викликати певний мікросервіс, користувачеві непотрібно заново передавати дані про ключ-доступу, він міститься на мікросервісі. Фреймворк, написано на мові високого рівня Java і підключається до проекту як бібліотека.

Висновки

Запропоноване вирішення проблеми авторизації користувачів у системах хмарних обчислень з мікросервісною архітектурою, шляхом розроблення фрейморку для роботи з різними типами аутентифікаційного механізму, шляхом побудови шару, який приймає всі запити. Цей фреймворк дає змогу програмісту менше часу для адміністрування різних типів ключів доступу.

Список літератури

1. Martin Fowler. Microservices [Електронний ресурс] — Режим доступу : https://martinfowler.com/articles/microservices.html Дата доступу: 09.10.2018
2. Офіційна документація KeyCloak. [Електронний ресурс]. – Режим доступу : https://www.keycloak.org/documentation.html Дата доступу: 09.10.2018
3. Офіційна документація JSON Web Token [Електронний ресурс]. – Режим доступу : https://jwt.io/introduction/ Дата доступу: 09.10.2018

ДОДАТОК Б

Застосування мікросервісної архітектури для побудови відмовостійкої хмарної систем

Акт впровадження

УКР.НТУУ “КПІ”.ТР32269_19М

Аркушів 2

2018